Imperial College London

Department of Mathematics

Combining non-exchangeable stochastic processes and dynamic graph VAEs for anomaly detection in computer networks

ALBERTO MAZZETTO

CID: 00892536

Supervised by F. SANNA PASSINO

September 3, 2023

Submitted in partial fulfilment of the requirements for the
MSc in Machine Learning and Data Science of
Imperial College London

The work contained in this thesis is my own work unless otherwise stated.

Signed: ALBERTO MAZZETTO Date: September 3, 2023

Abstract

As cybersecurity defence becomes increasingly challenging, statistically principled anomaly detection methods become increasingly relevant. In this project, we extend a local anomaly detection methodology, based on Bayesian non-parametric stochastic processes, by dropping the exchangeability assumption, a condition that is almost certainly not verified in practice. We seek to do so by accounting for inter-arrival times, in three different ways: by modelling the time sequence of events with a stochastic process, by considering a streaming version of the processes, or by including dependence on inter-arrival times explicitly within the process, by means of a distance-dependent Chinese restaurant process. The proposed methods are compared and contrasted both on a synthetic dataset, with simulated intrusion, and an open-access dataset of Windows authentications within Los Alamos National Laboratory's enterprise network. In cybersecurity applications, special attention has to be posed on reducing false alarms, to avoid overloading security teams; the second core contribution in this work is a methodology to verify if unexpected communication patterns are found in the neighbourhood of potentially anomalous nodes, as flagged by the non-exchangeable stochastic processes, and upgrade their priority accordingly. One key aspect of the proposed procedure is that normal or unexpected patterns are discriminated on the basis of what communication patterns are common within the entire network, rather than just in the neighbourhood under investigation, seeking to introduce global information to improve the overall anomaly ranking. To give an example of the intuition behind this idea, suppose that an anomaly-free computer changes its destination of use; its local behaviour would change significantly, and the local anomaly detection would find high anomaly scores, but perhaps its communication patterns would be quite similar to the ones of computers with the same intended use, hence, the global part of anomaly detection would mitigate its anomaly ranking. The modelling approach chosen to learn the normal patterns within the network is a dynamic graph variational auto-encoder: we learn a latent probabilistic space for node embeddings, having in such way the flexibility to describe sub-graphs with varying node set cardinality, and allowing for the calculation of the probability of an unseen sub-graph given the model, with Monte Carlo sampling. It is indeed by calculating the p-value of an unseen dynamic graph, given the model trained on the entire network, that we reorder the anomaly scores provided by the non-exchangeable stochastic processes.

Acknowledgements

I would like to thank Prof. G. Nason, Prof. N. Heard and the entire course staff for supporting, directing and teaching this MSc in Data Science and Machine Learning with dedication and passion.
I would like to acknowledge my supervisor Prof. F. Sanna Passino, for his teachings and for the directions and insights that were fundamental for the development of this project.
I am also very grateful to J. Neil, for the time dedicated to conversations on cybersecurity that were extremely inspirational for me and my work.
Many thanks to my tutor, Z. Varty, for the regular and useful exchanges that we had throughout this journey.
A final special thanks should go to my closest course mates, with a special mention to F. Nucera, for the useful discussions, brainstorming chats and opportunities to exchange ideas and views.

Contents

1. Background

This chapter starts with an introduction to cybersecurity, the motivating context for this project. Nonetheless, the techniques presented throughout are broadly applicable to any unstructured dataset representable under a dynamic graph paradigm. The chapter proceeds by outlining the contributions of this work, supporting literature and alternative options that were considered but not explored, and could be inspiring for future extensions.

1.1. Cybersecurity

Scale, complexity and impact of **cybersecurity attacks** have been increasing in recent years. This is somehow linked to the ongoing digitalisation, that opens to new opportunities as well as threats, the diffusion of home working during and after the COVID-19 pandemic, which created potentially more liabilities, and the Russian-Ukrainian cyberwarfare [37, 38]. 28% of small and medium-sized European businesses suffered from cybercrime in 2021 and the estimated global **economic cost** in 2020 is 5.5 trillion Euro, double than in 2015 [38]. There are other costs of cyberattacks, such as **democratic** and **peace and stability related**, which are linked to disinformation attacks (fake news, deep fakes) as well as attacks - such as malware and denial of service - which undermine the functioning of institutions and governmental agencies [37]. The sectors more at risk are the ones that rely the most on digital technologies, essential services and critical sectors, ranging from health systems to power grids and water supplies [37].

To counteract the fast-paced evolution of cybersecurity threats, from at least the last decade the focus has shifted towards integrating rule-based security measures (also known as signature-based detections) with **statistically grounded methods**, aiming at recognizing anomalous computers and patterns in the computer network. Perhaps one of the most natural mathematical representations of a computer network is as a graph.

Definition 1.1.1 (Computer network graph)**.** A **computer network graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is given by a node set $\mathcal{V} = \{x_i\}$, comprising all machines in the network, and an edge set $\mathcal{E} = \{(x_i, x_j) \in \mathcal{V} \times \mathcal{V} : N(x_i, x_j) > 0\}$, where $N(x_i, x_j)$ is a counting process of connection requests between nodes x_i and x_j.

As the number of computers and connections varies over time, we need to account for such dynamic behaviour. If the data were collected at discrete timestamps $t \in \mathcal{T}$, then the graph at each time-step can be defined like in 1.1.1, with cardinalities of node and edge sets potentially different at each time t. However, it is common to aggregate the

data, as the graph at each single time-step might be too sparse. In general, aggregation is necessary with continuous time-stamps.

Definition 1.1.2 (Aggregated dynamic graph)**.** An **aggregated dynamic graph** requires partitioning of the time dimension in - usually equal - intervals of size δ:

$$I_i = \{t : t \in [(i-1)\delta, i\delta], \delta > 0\}, \text{ with } i \in \mathbb{Z}^+.$$

We can then define the graphs $\mathcal{G}_{I_i} = (\mathcal{V}_{I_i}, \mathcal{E}_{I_i})$ with node sets:

$$\mathcal{V}_{I_i} = \{x_j : \exists t \in I_i \text{ for which } x_j \text{ exists}\},$$

and edge set:

$$\mathcal{E}_{I_i} = \{(x_j, x_k) \in \mathcal{V}_{I_i} \times \mathcal{V}_{I_i} : N_{I_i}(x_j, x_k) > 0\}.$$

Anomaly detection algorithms seek to describe the "normal" course of the graph's history and spot unexpected patterns that hint at possible intrusions. In most types of cyber threats, there is a factor that makes anomaly detection somehow feasible: sensitive computers are usually on premise and well fortified from the outside world, hence, intruders often have no alternative other than braking into more liable machines and move laterally within the enterprise network until the target information is found [7, 8]. Such traversing is very likely to generate a series of unexpected connections which cyber defence software try and identify. There are two important **challenges** in cybersecurity anomaly detection that should be kept in mind when developing new algorithms. The first is the necessity to process data and identify anomalies virtually in **real time**, before any sensitive machine gets compromised. To give an idea of the reaction times, we cite the 1-10-60 golden rule by a global cybersecurity provider [4]: there is roughly one minute to detect a potential intrusion, 10 minutes to investigate it and 60 minutes to resolve it. The second challenge has to do with **false alarms**: as the activity within the network is intense, algorithms with an excessive number of false detections would unnecessarily overload security teams, and such event should be reduced. These challenges motivate some choices made in this project: as we will see shortly, we work with a local anomaly detection algorithm, as opposed to global algorithms, on the basis that it is better suited for distributed, fast calculations, and follow the first anomaly detection with a second, more global, anomaly detection designed to reduce the chances of false alarms.

1.2. Anomaly Detection in Dynamic Graphs

There are two main approaches to dynamic graph modelling, **global** and **local**. **Global** methods [2, 27, 6, 13, 24, 23] aim at uncovering a latent structure - or space - from which the graph itself could have been generated; this can be done by seeking lower-dimensional representations of the adjacency matrix, or by relating its entries to node-specific vectors or feature vectors. As we will be concerned with local methods, more details on global

approaches can be found in Appendix A. **Local** models [35, 12, 28, 11, 17], on the other hand, focus on modelling activity at the node or edge level. Such approaches draw from time history modelling theories, and have the type of chosen process as main differentiator, whether that is a Markov chain [35], a Bayesian non-parametric approach [12, 28], a Poisson process for the number of authentications [11] or a Hawkes process [17]. The strength of global methods resides in the fact that information sharing across the network is embedded in the modelling approach, while local methods are usually simpler, less computationally expensive and more easily scalable than global methods, hence possibly better suited for cybersecurity applications in large networks. Amongst local methods, Bayesian non-parametric approaches proved very powerful, as they can handle a potentially infinite number of machines, incorporate prior knowledge and can naturally deal with cold starts. These are the reasons that drove us towards reviewing in detail and extending [12, 28].

[12] model Windows authentication sequences of the sort `source;destination` by postulating an independent Dirichlet processes for each destination node: if $\mathcal{X}$ and $\mathcal{Y}$ are the sets of source and destination computers respectively, the cited work seeks to calculate the predictive p-value of an observed $(x, y) \in \mathcal{X} \times \mathcal{Y}$ by integrating probabilities of the sort $p(x|y)$ assuming that $\mathbb{P}_{\mathcal{X}|y} \overset{d}{\sim} DP(\alpha_y \cdot \mathbb{P}_0)$. The Dirichlet process is a Bayesian non-parametric model for the probability distribution over probability distributions, defined as follows:

Definition 1.2.1 (Stick-breaking formulation of the Dirichlet Process)**.** We say that the probability distribution over the set of probability distributions $\mathbb{P} \sim DP(\alpha \cdot \mathbb{P}_0)$ is a Dirichlet stochastic process if the corresponding mass function is:

$$p(x) = \sum_{j=1}^{\infty} w_j \mathbb{I}_{\theta_j}(x),$$

where the atoms of mass are independently drawn from the base distribution $\theta_1, \theta_2, \cdots \sim \mathbb{P}_0$ and w_j are drawn from a stick-breaking process $w_1, w_2, \cdots \sim GEM(\alpha)$, with $\gamma_j \sim Beta(1, \alpha)$. This means that $\sum_{i=1}^{\infty} w_i = 1$ and $w_i = \gamma_i \prod_{k=1}^{i-1}(1 - \gamma_k)$ with γ_is random variables in $[0, 1]$.

Figure 1.1 shows two distributions drawn from a Dirichlet Process, with geometric base measure. No matter whether the base measure is discrete or not, the draws from a Dirichlet process are almost surely discrete. It is possible to note how the bigger α, the closest the distribution resembles the base measure. The Dirichlet process is a conjugate model, and after observing the sequence $(x_1, y), \cdots, (x_n, y)$ with destination y, the posterior distribution is again a Dirichlet process:

$$\mathbb{P}_{\mathcal{X}|\mathbf{x}_n, y} \overset{d}{\sim} DP\left(\alpha_y \cdot \mathbb{P}_0 + \sum_{i=1}^{n} \delta_{x_i}\right),$$

and the predictive probability for the next element in the sequence has a simple form:

$$p(x|\mathbf{x}_n, y) = \frac{\alpha_y}{\alpha_y + n}\mathbb{P}_0(x) + \frac{1}{\alpha_y + n}\sum_{i=1}^{n}\mathbb{I}_x(x_i),$$

which is used to calculate a predictive p-value. In the paper, the chosen base measure is the uniform distribution, hence uniform surprise a priori, and α is chosen to be proportional to the nodes in-degree, as computers which usually receive a great number of connection requests are likely to be less "surprised" by a new connection. After m

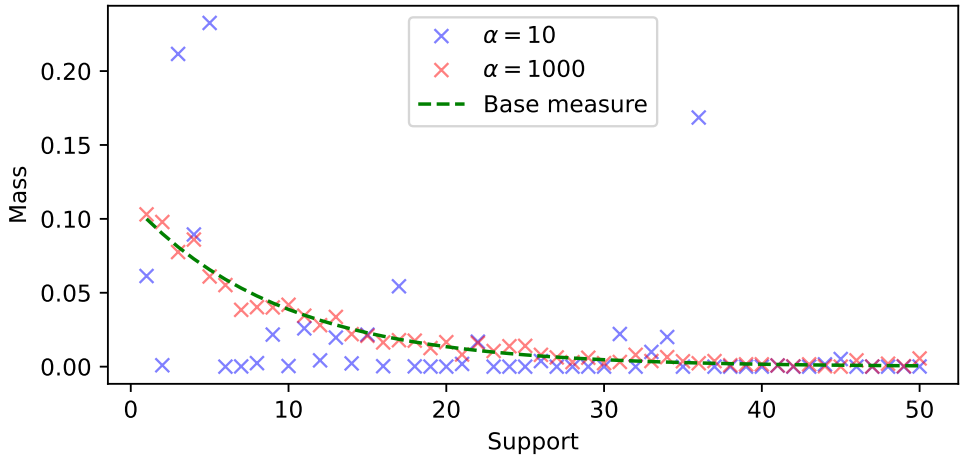

Figure 1.1.: Draws from a Dirichlet Process

connections, the p-values for each edge are combined with a minimum p-value combiner [1], and finally the edge p-values are combined to generate source node "surprise" scores.

A later work [28] extends upon similar ideas by introducing several improvements. The first is the use of a Pitman-Yor process, which allows for power-laws in the degree distribution of the nodes, a condition often verified in practice. Consider an infinitely exchangeable sequence of nodes $x_1, x_2, \cdots \in \mathcal{V}$. After observing n events, of which K_n are unique, the distribution of observed nodes is a Pitman-Yor process $\mathcal{PY}(\alpha, d, \mathbb{P}_0)$ if the predictive distribution for the next observed node is:

$$p(x|\mathbf{x}_n) = \frac{\alpha + dK_n}{\alpha + n}\mathbb{P}_0(x) + \sum_{j=1}^{K_n}\frac{N_{jn} - d}{\alpha + n}\delta_{x_j^*}(x), \tag{1.1}$$

where $\mathbb{P}_0(\cdot)$ is a base probability distribution on $\mathcal{V}$, $d \in [0, 1)$ is a discount parameter, $\alpha > -d$ is a strength parameter, $N_{jn} = \sum_{i=1}^{n}\delta_{x_i}(x_j^*)$ is the number of observations for each unique value x_j^* after n observations. For $d = 0$ the Dirichlet process is recovered; the Pitman-Yor process can be seen as a two-parameter version of the Dirichlet process

[1] To consider the most extreme connection

with a parameter to further discount infrequent observations. As explained in [28], considering the two-parameters model admits power-laws in the number of corrections on each edge and - for careful choices of the parameters - power-laws [2] for the degree distribution of the nodes, conditions that are often verified in computer networks. A common metaphor gives more intuition about the process; note that Equation 1.1 is composed of two terms, the first is the probability of drawing a new x from the base measure:

$$p(x : x \notin \{x_j^*\}_{j=1}^{K_n} | \mathbf{x}_n) = \frac{\alpha + dK_n}{\alpha + n},$$

while the probability of drawing x of a kind already drawn is:

$$p(x : x = x_j^* | \mathbf{x}_n) = \frac{N_{jn} - d}{\alpha + n}.$$

It is necessary to highlight immediately here that the notation $p(x : x \notin \{x_j^*\}_{j=1}^{K_n} | \mathbf{x}_n)$ is valid only for continuous base measures, for which the probability of drawing the same sample more than once is zero, whereas for discrete base distributions a new sample drawn from the base measure could well be in $\{x_j^*\}_{j=1}^{K_n}$. Let us stick with the continuous case to explain the metaphor: if each x is a client entering a restaurant, then with probability $p(x : x \notin \{x_j^*\}_{j=1}^{K_n} | \mathbf{x}_n)$ she would sit at a new, empty table, while with probability $p(x : x = x_j^* | \mathbf{x}_n)$ she would sit at a table occupied by customers of type x_j^*. The second improvement proposed in the paper, is a hierarchical process for destination and source computers, where not only is $\mathbb{P}_{\mathcal{X}|y} \overset{d}{\sim} PY(\alpha_y, d_y, \mathbb{P}_0)$ but also $\mathbb{P}_{\mathcal{Y}} \overset{d}{\sim} PY(\alpha_0, d_0, \mathbb{P}_0)$. Another important contribution in [28] is in the estimation of the intensity and discount parameters of the Pitman-Yor process by an extended method of moments, that will be detailed later. Finally, the combination of p-values to obtain source node p-values is conceptually analogous to [12], but more p-value aggregation methods were tested.

The **first idea developed in this project** is to consider further extensions to the above-mentioned works, by **breaking the complete exchangeability assumption**. In fact, the Pitman-Yor process (and the Dirichlet process likewise) has such property: given the sequence of events x_i with $i = 0, 1, \cdots, n$, no matter the ordering of such events, the joint probability under the Pitman-Yor process is unchanged:

$$P(x_1, \cdots, x_n) = P(x_{\pi(1)}, \cdots, x_{\pi(n)}),$$

where $\pi(\cdot)$ is a permutation of the events. Despite such a strong assumption, the Pitman-Yor process proved very powerful also when tested on real data; however, in practice, there are relationships between subsequent events which almost surely do not guarantee such property, and from here the idea to drop this assumption. We provide details on how we wish to drop exchangeability in the next section.

The **second contribution** in this project is on an anomaly detection methodology to follow the previous local anomaly detection, which integrates more global information to reduce false alarms. A similar concept is found in [35], where local anomaly p-values are

[2] Recall that $\kappa \sim PL(\gamma)$ if $p(\kappa) \propto \kappa^{-\gamma}$, $\kappa \geq 1$, $\gamma > 1$

combined for the connected components of anomalous network sub-graphs, to confirm or downgrade the level of anomaly by integrating information from neighbouring nodes. We will refer to this part as **second part of anomaly detection**, as opposed to the previous local **first part of anomaly detection**. The novel proposal in this project is to use a model-based approach, by training a dynamic graph model to capture the "normal" behaviour of opportunely defined dynamic sub-graphs in the neighbourhood of any node, and calculating the p-value of potentially anomalous sub-graphs given such model. The link to the previous, local anomaly detection is that its anomaly score is used as node feature and its p-values are combined with the output of the second part of anomaly detection, to produce an ultimate anomaly ranking. The intuition here is to try and exploit the fact that an intrusion is almost surely followed by lateral movement within the network which would create an unexpected pattern of communication. If the anomalous patterns were known a priory, one could use pattern matching algorithms for dynamic graphs [40, 3]. However, we believe that a model-based approach is more flexible to accommodate for unforeseen patterns, as the model is trained on data and does not require prior knowledge of the anomalous patterns, which should make it more robust. In the next section we dive into the details of the chosen modelling approach.

1.3. Contributions

As anticipated, in the first part of anomaly detection we seek to drop the exchangeability assumption underlying [12, 28]. The Bayesian modelling approaches seen before do not use the timestamp information of the marked point process $\mathcal{P} = \{(t_i, x_i, y_i), i \in \mathbb{N}\}$, if not to order the observations chronologically. We have thought of three ways to use the timestamp information and make the process no longer exchangeable. The first option is to **model the arrival times with a stochastic point process**: if the hierarchical model in [28] models the probability of a connection as $p(x, y) = p(x|y)p(y)$, we would extend by modelling $p(x, y, t) = p(t|x, y)p(y|x)p(y)$. We will explain later that such task is very challenging, as guaranteeing consistency between the time and Pitman-Yor processes is not trivial, as the former imposes a spacing between events which must be compatible with the ordering given by the latter. We will resort to the simple Poisson Point Process, as it disposes of a partitioning property. Another, simpler way to drop the exchangeability assumption is to consider a **streaming version of the process**, where instead of calculating the probability of a new event based on the entire history, $p(x_{n+1}|\{x_i\}_{i=1}^n)$, we consider $p(x_{n+1}|\{x_i : t_{n+1} - t_i < T\})$ where T is the window of the streaming process. It is also possible to model more explicit relationships between the data, for example in the third and last approach used in this study, which is the so called **Distance Dependent Chinese Restaurant Process** [5]; the details will be explained in Chapter 2 after introducing the Chinese Restaurant metaphor, for now it suffices to consider that the implementation of such process introduces an explicit time dependency between the data, and the parameter estimation can take advantage of the dissertation in [28].

We mention here other two possibilities that were considered but not implemented,

and might be suitable for future extensions. Using a forgetting factors version of the Dirichlet process as described in [25], where the principle of forgetting introduced in [16] is applied to down-weight older observations, could be a valid alternative to the streaming version presented in this work. Moreover, other formulations of dependency in the Dirichlet process exist, for example the Dependent Dirichlet Process used for time series modelling [20, 32, 19, 9]. This framework is used in [9] to model a stock price evolution including volatility, and could possibly be applied to cybersecurity where "volatility" changes between day and night, for example. The main concern with such and similar models is that complexity and parameter estimation might be too challenging in a cybersecurity application.

Given the potentially anomalous nodes from the first part of anomaly detection, the novel idea in the second part of anomaly detection is to construct a dynamic graph in a carefully chosen neighbourhood of each node, and calculate its p-value given a model that describes, and expects, conditions not affected by external intrusions. Given the scale and complexity of the problem at hand, the model of choice is a deep learning model, namely, a **dynamic graph variational auto-encoder**. Deep learning variational auto-encoders learn a probabilistic latent variable $\mathbf{z}$ that unveils hidden structure in the data. These are likelihood-based models, for which each observation x is associated with a likelihood function. The marginal likelihood, or model evidence, of a single data-point $x \in \mathbb{R}^d$ is:

$$p_\theta(x) = \int p_\theta(\mathbf{z}) p_\theta(x|\mathbf{z}) d\mathbf{z},$$

where $p_\theta(\mathbf{z})$ expresses the prior on $\mathbf{z}$ and $p_\theta(x|\mathbf{z})$ is the probability of an observation given $\mathbf{z}$, with θ a set of parameters. The variational auto-encoder, trained on "normal" graphs, would learn the model parameters θ by means of an ELBO approximation, as this integral is intractable. Although the prime purpose of variational auto-encoders is not to evaluate the likelihood of new samples conditional on the model, it is possible to calculate such quantity integrating over the hidden variable using Monte Carlo methods. Once the likelihood for each potentially anomalous sub-graph $p(\mathcal{G}_i)$ is available, this information can be used to calculate a likelihood score or a p-value for each potential anomaly. The encoder network of such model would include a Graph Neural Network, as the inputs are graphs and permutation invariance must be fulfilled, with scores from the previous phase of anomaly detection as node features. In this work we use ideas from [10] to build a variational auto-encoder for dynamic graphs. The modalities of selection of the sub-graphs and training of the model will be discussed later in a dedicated section. To conclude, we would like to mention that another alternative model, amongst others, would be normalising flows, where a simple and tractable probability distribution is morphed via invertible transformations to produce a complex probability distribution that can better represent the data. There are examples of such applications both for static [18] and dynamic [39] graphs. The likelihood of a sample given the model is trivial to calculate with the change of variable formula, although getting a good model architecture to approximate the data in the first place is quite challenging.

2. Methods

We will be working with the most general type of computer network authentication data, interpreted as a marked point process $(\mathcal{T}, \mathcal{X}, \mathcal{Y}) = \{(T_i, X_i, Y_i)\}_{i\geq 1}$ where T_i is the **time** random variable defined on $\mathbb{R}^+$ and the marks $(X_i, Y_i) \in \mathcal{V} \times \mathcal{V}$ [1] are the **link** random variables. The chapter progresses by introducing the methodologies adopted for both the first and second phase of anomaly detection, focusing on the novel contributions of this work.

2.1. First Part of Anomaly Detection

Analogously to [28] we will be concerned with a hierarchical Bayesian non-parametric model, $\mathcal{BNP}$, for the sequence of realised links. The probability of a link $p(x, y)$ could be decomposed, using Bayes' theorem, either as $p(x, y) = p(y|x)p(x)$ or $p(x, y) = p(x|y)p(y)$. When x and y are interpreted as source and destination computers of a communication, the choice is between selecting either of the two as the view-point to seek anomalies. As argued in [12], given that any computer in the network is likely to send authorisation requests (intuitively, every employee's personal computer is likely to communicate with the authentication server at least once a day), while fewer would be recipients, it seems convenient to look for anomalies by taking the recipient's viewpoint. The model is formulated as follows:

$$\begin{aligned} \mathbb{P}_{\mathcal{Y}} &\overset{d}{\sim} \mathcal{BNP}(\theta_0, \mathbb{P}_0), \\ \mathbb{P}_{\mathcal{X}|y_i} &\overset{d}{\sim} \mathcal{BNP}(\theta_{y_i}, \mathbb{P}_0), \end{aligned} \tag{2.2}$$

where the probability mass function $\mathbb{P}_{\mathcal{Y}}$ regards the destination sequence, while $\mathbb{P}_{\mathcal{X}|y_i}$ regards the source sequence given the realisation y_i of the stochastic variable $\mathcal{Y}$. Both PMFs have support over the nodes set $\mathcal{V}$, and so does the base measure $\mathbb{P}_0$ which is assumed here to be the same for both processes and will be chosen to be $Unif(\mathcal{V})$, as this is the most general assumption possible. Final mentioning of the parameters θ_0, for the destination process, and θ_{y_i}, for source processes given y_i. In the following, we will explain and detail the choice of the process that here was generally denoted as $\mathcal{BNP}$. The structure for all models will follow Equation 2.2 as far as the **mark** part of the marked point process is concerned. In one of the models, as anticipated, we will take a step further and also model the point process $\mathcal{T} \sim \mathcal{PP}(\lambda)$, where λ is a parameter. The scenarios are:

1. Pitman-Yor process and Dirichlet process as a special case, replication of [28, 12];

[1] As before, $\mathcal{V}$ denotes the node set

2. Pitman-Yor process for the marks and Poisson point process for times;

3. Streaming version of Pitman-Yor point process;

4. Distance dependent Chinese restaurant process.

2.1.1. Pitman-Yor Hierarchical Model

The benchmark models are the same like in [28] and a hierarchical extension of [12] with parameter estimation done with the method of moments like in [28]. As the Dirichlet process is a special case of the Pitman-Yor process, we will be concerned only with the latter in this dissertation. Referring to 2.2, we set $\mathcal{BNP}(\theta_0, \mathbb{P}_0) = PY(\alpha_0, d_0, \mathbb{P}_0)$ and $\mathcal{BNP}(\theta_{y_i}, \mathbb{P}_0) = PY(\alpha_{y_i}, d_{y_i}, \mathbb{P}_0)$. In order to calculate a p-value to quantify the unexpectedness of the next observation, we consider the fact that the predictive probability of a new connection request is like in Equation 1.1:

$$p(x|\mathbf{x}) = \frac{(\alpha + d \cdot K_n)p_0(x) + \sum_{i=1}^{K_n}(N_{in} - d)\delta_{x_i^*}(x)}{\alpha + n} = \frac{\alpha_x^*}{\alpha^*}.$$

If the realised value is x_{n+1}, then the p-value associated to it is calculated as the predictive probability of observing a node as extreme as x_{n+1}:

$$\mathbf{p}_{n+1}^{RIGHT} = \sum_{x \in \mathcal{V}: p(x|\mathbf{x}) \leq p(x_{n+1}|\mathbf{x})} p(x|\mathbf{x}) \tag{2.3}$$

As explained in [28], for discrete probability distributions using mid-p-values performs a lot better. The mid-p-value is defined as:

$$\mathbf{p}_{n+1} = \frac{1}{2}\left(\mathbf{p}_{n+1}^{LEFT} + \mathbf{p}_{n+1}^{RIGHT}\right),$$

where $\mathbf{p}_{n+1}^{RIGHT}$ is defined as above and $\mathbf{p}_{n+1}^{LEFT}$ similarly, but considering only smaller p-values $p(x|\mathbf{x}) < p(x_{n+1}|\mathbf{x})$. There are some mathematical simplifications that make this calculation extremely tractable, the details of which can be found in Appendix B.

2.1.2. Poisson Point Process

The first extension presented in this work is concerned with calculating a p-value for times as well. The PY hierarchical process poses an expectation over the ordering of events, but otherwise no importance is given to inter-arrival times. The question now is whether the point process should be considered at the edge or global level. Considering the process at the edge level would mean considering $p(t, x, y) = p(t|x, y)p(x|y)p(y)$, whereas at the global level $p(t, x, y) = p(t)p(x|y)p(y)$. p-values can still be obtained by integrating over the entire space, or, like in our case, approximated with an aggregation of p-values. The other question is whether we should consider different parameters for each edge λ_{xy} or not. The point here is that there is not complete independence between the time process(es) and PY sequences, given that both determine an ordering

of the observations. We will come back to this later, after introducing some notions on the Poisson Point Process and a property that is key to guaranteeing the desirable congruence between the ordering processes.

Definition 2.1.1. (Renewal Process) Continuous-time stochastic process in which the increments are independent from one another. The stochastic process is denoted as $\mathcal{N} = \{N(t)\}_{t\geq 0}$ with $N(t) = \max\{n : T_n \leq t\}$, $T_0 = 0$, $T_n = X_1 + \cdots + X_n$, $n \geq 1$ with $\{X_i\}$ independent, identically distributed, non-negative random variables.

Definition 2.1.2. (Poisson Process) Renewal process with inter-arrival times modelled like $X_i \sim Exponential(\lambda)$, and for which $N(t) \sim Poisson(\Lambda) = Poisson(\lambda t)$
The non-homogeneous Poisson Process has intensity $\lambda(\cdot)$ that depends on time.

The Poisson Process has a nice partitioning property [31]:

Theorem 2.1.3. (Poisson Process Partitioning) If $N \sim Pois(\lambda)$, and each event is categorised into two categories with probability p and $1-p$ respectively, then each sequence conditioned upon its category follows a Poisson process with $N_1 \sim Pois(p\lambda)$ and $N_2 \sim Pois((1-p)\lambda)$

This is very useful in our application, because if we were to assume that the marked point process $(\mathcal{T}, \mathcal{X}, \mathcal{Y})$ was Poisson with intensity λ or $\lambda(\cdot)$, then we could use the partitioning dictated by the Pitman-Yor hierarchical process to derive the Poisson process at destination node level or link level. If the global counting process is:

$$N \sim Pois\left(\int_0^t \lambda(s)ds\right),$$

then for each destination we have that:

$$N_y(t) \sim Pois\left(\int_0^t \lambda(s)p\left(y|\{y_i \in \mathbf{y}_n : t_i \leq s\}\right)ds\right),$$

and for each link (x, y):

$$N_{xy}(t) \sim Pois\left(\int_0^t \lambda(s)p\left(x, y|\{x_i \in \mathbf{x}_n : t_i \leq s\}, \{y_i \in \mathbf{y}_n : t_i \leq s\}\right)ds\right).$$

The positive of this approach is that it guarantees consistency between all the processes involved. The downside is that it requires to calculate integrals with PY probabilities as arguments, which requires calculating such probabilities for all links in the network and, moreover, store many of them to be able to calculate the integral. Under this model, whenever a new sample is observed, it is possible to calculate the p-value associated with the Poisson point process using the following fact:

$$N(t+s) - N(t) \sim Pois\left(\int_t^{t+s} \lambda(l)dl\right). \tag{2.4}$$

Whenever a new observation is recorded between $[t_n, t_{n+1}]$ we calculate the arrival time p-value as the mid p-value given by the Cumulative Distribution Function (CDF) of a Poisson distribution with rate $\int_{t_n}^{t_{n+1}} \lambda(l)dl$:

$$\mathbf{p}_{n+1} = 0.5(CDF(0) + CDF(1)),$$

where $CDF(0)$ is the probability of observing no events, $CDF(1)$ is the probability of observing no or one event. The calculation of $p(y)$ and $p(x|y)$ and the associated p-values is performed like in the previous case.

2.1.3. Streaming Pitman-Yor

A simple way to drop the complete exchangeability assumption in the PY process is to consider a **sliding window** version [21], giving more relevance to recent data points. Given a fixed window span $\delta \in \mathbb{R}^+$, of the entire time sequence registered up to the n-th observation, $\{t_1, \cdots, t_n\}$, at current time t we consider only only $\{t_j : t - t_j < \delta\}$ for parameter estimation or, in our case, p-value calculation. We have not yet discussed parameter estimation in our model, which we will see addressed with a generalised method of moments, however, with this streaming version rather than affecting the parameter estimation we use a sliding window to determine what past observations contribute to the calculation of Equation 1.1. Given the marked point process $\{\mathcal{T}, \mathcal{X}\}$ with realisations $\{(t_i, x_i)\}_{i=1}^n$, we seek to calculate the probability of a new observation (t, x) as:

$$p(x|\{x_i : t - t_i < \delta\}).$$

An important aspect of streaming algorithms is calculating the memory requirements, given the necessity to store data covering the entire time window. In our application, for the p-values calculation in Equation B.20 and Equation B.21 we have to store only and exclusively counts $n(x_j^*)$ for each unique x_j^*. In the sliding window context, whenever (t, x) is observed all counts have to be updated according to:

$$n_t(x_j^*) = n_{t_n}(x_j^*) - |\{x_i : x_i = x_j^*, t - t_i > \delta\}_{i=1}^n| + \mathbb{I}_{x_i^*}(x),$$

an expression that can be written in recursive form, reducing drastically the memory requirement that becomes of the order of the number of observations between current time t and $t - \delta$. If we define $s(t) \in \mathbb{N}$ the index of the sequence such that $t - t_{s(t)} < \delta$ and $t - t_{s(t)-1} > \delta$ - hence, the index that separates between obsolete and relevant observations - we can write:

$$n_t(x_j^*) = n_{t_n}(x_j^*) - |\{x_i\}_{i=s(t_n)}^{s(t)}| + \mathbb{I}_{x_i^*}(x). \tag{2.5}$$

Note that it is in recursive form because t_n is the last element of the sequence of recorded observations, hence the previous with respect to current time t. With this formulation, relevance is given exclusively to observations in the sliding window, with an abrupt separation to older observations and equal importance for all observations in the window. The approach presented next could be seen as a smoother alternative.

2.1.4. Distance Dependent Chinese Restaurant Process

The third and last contribution concerning the first part of anomaly detection has to do with applying the distance dependent Chinese restaurant process to our case study. In [5], the authors introduce the **distance dependent Chinese restaurant process**, which allows to include various types of dependencies between the observations, which in our application is a time dependency. Stating, for example, that more recent observation are more likely to reappear than older observations, is in itself a negation of the exchangeability principle (as the ordering of the observations would matter to total probability calculation) and might well be the case in practice. The model proposed in the article is a Chinese restaurant process where the stochastic seating assignment is dependent on the arrival distance between customers.

Definition 2.1.4. (Distance dependent Chinese restaurant process) Denote with x_k^* the group of customers to which customer x_i is assigned, with d_{ij} the distance between customer i and j and let $f(\cdot)$ be a decay function. The distance dependent CRP draws the customer x_i assignment conditioned upon the distance measurements like:

$$p(x_i = x_k^* | \mathbf{d}, \alpha) \propto \begin{cases} \alpha & x_k^* \neq x_j \forall j \\ f(d_{ij}) & x_k^* = x_j \end{cases} \tag{2.6}$$

The first important aspect to note is that the shift has moved from tables, in the standard CRP, to customers: we no longer speak of table assignments but customer group assignments which will then result in such groups of customers sitting at different tables. Equation 2.1.4, in turn, is expressing the fact that:

- With probability proportional to α customer x_i is assigned a new label x_k^* different from the labels of all other clients, hence would be sitting by herself until any other client might be assigned to the same group;
- With probability proportional to $f(d_{ij})$ customer x_i is sitting with customer x_j and hence inheriting its label x_k^*.

To help with the comprehension, Figure 2.1 and Figure 2.2 picture the standard and distance dependent CRP respectively. In our application we work in the sequential CRP case, as future observations are yet to be observed. One important property of distance-dependent CRP is that they are not **marginally invariant**: removing one customer, in general does not leave the partition distribution over remaining customers unchanged, and the only case when marginal invariance is achieved is with the traditional CRP which is for $f(\cdot) = 1$. This property differentiates the choice of this process from the random-measure models [9, 19], where marginal invariance is satisfied.

Now that the key concepts are introduced, we are left with two important tasks: deciding what proportionality constants to have in Equation 2.1.4 and what decay function to use. Regrading the first point, the choice has been to use the same proportionality for unseen observations like in the Dirichlet process, in order to be able to use the same

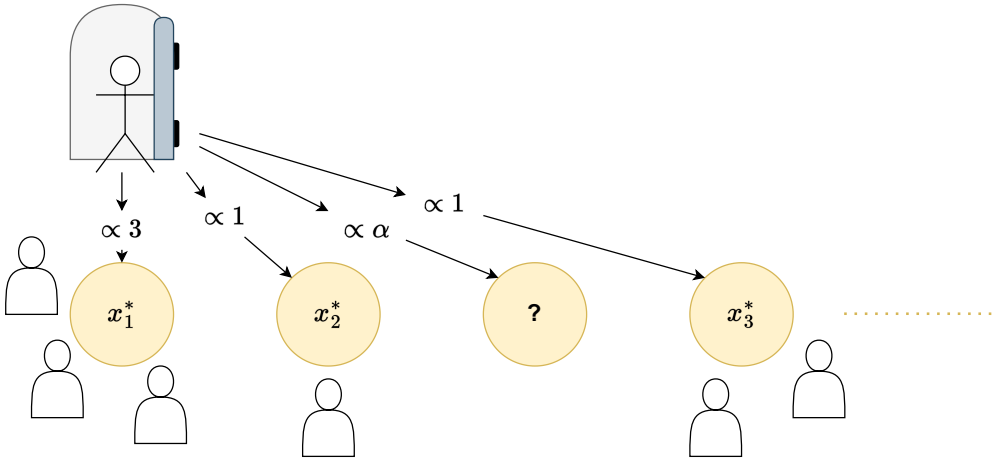

Figure 2.1.: Exemplification of the standard Chinese Restaurant Process

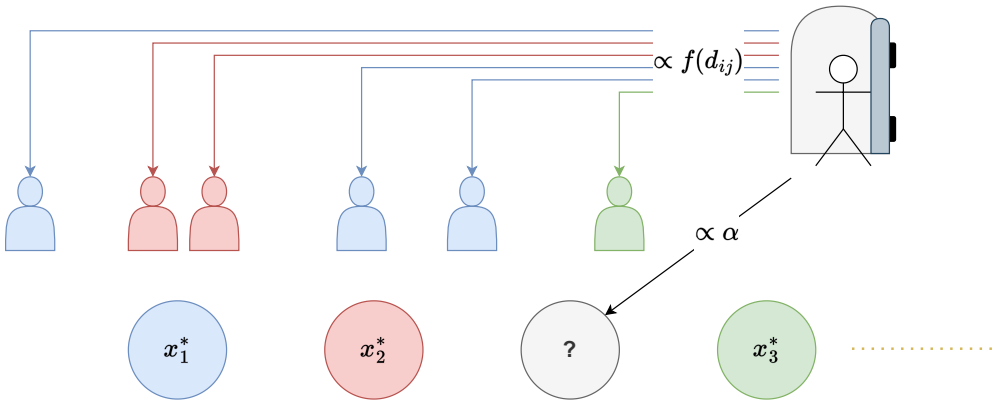

Figure 2.2.: Exemplification of the distance dependent Chinese Restaurant Process

parameter estimation techniques that will be explained in the following chapter. The constant for the other term, follows from the requirement for a probability measure that must integrate to one:

$$p(x|\mathbf{x}_n) = \frac{1}{\alpha + n}\left[\alpha p_0(x) + \frac{n\sum_{i=1}^{n} f(d(x_i, x))\mathbb{I}_x(x_i)}{\sum_{i=1}^{n} f(d(x_i, x))}\right], \tag{2.7}$$

where $d(\cdot,\cdot)$ is a chosen distance measure. We choose to have $d(x_i, x_j) = t_j - t_i$. A convenient choice for the decay function is exponential, as this would simplify the calculations in a streaming context:

$$f(\gamma) = e^{-\gamma/\beta}.$$

This allows the following simplifications in the second term of Equation 2.7:

$$\frac{\sum_{i=1}^{n} f(d(x_i, x))\mathbb{I}_x(x_i)}{\sum_{i=1}^{n} f(d(x_i, x))} = \frac{\sum_{i=1}^{n} \exp(-(t - t_i)/\beta)\mathbb{I}_x(x_i)}{\sum_{i=1}^{n} \exp(-(t - t_i)/\beta)} = \frac{\sum_{i=1}^{n} \exp(t_i/\beta)\mathbb{I}_x(x_i)}{\sum_{i=1}^{n} \exp(t_i/\beta)}.$$

Note that having no dependence on the current time t allows to store the sequential calculations of $\sum_{i=1}^{n} \exp(t_i/\beta)\mathbb{I}_x(x_i)$ without having to keep into memory all observations. One extra note on the fact that we need to be careful with avoiding overflow with the calculation of the sum of exponentials, especially when the process runs for a long time. With this choice of normalising constants we are entitled to use, for the p-values, similar calculations to the ones that led to Equation B.20 and Equation B.21, with the necessary adjustments.

2.1.5. Parameters Estimation

In the models presented so far, we are concerned with the estimation of the intensity parameter α for the distance-dependent CRp, with the intensity α and discount d parameters for the PY process, and with the estimation of the intensity parameter $\lambda(t)$ for the inhomogeneous Poisson process.

The choices that were made for the distance-dependent CRP make parameter estimation analogous to the case of the Dirichlet process. Indeed, both processes can be written as:

$$p(x|\mathbf{x}_n) = \frac{\alpha}{\alpha+n} p_0(x) + \frac{1}{\alpha+n} c(x, \mathbf{x}_n).$$

If $p_0(\cdot)$ is continuous, then the probability of drawing an unseen value for each $i = 1, \cdots, n$ is $\frac{\alpha}{\alpha+i-1}$. Consequently, the number of clusters K_n after n draws is [34, 1]:

$$\mathbb{E}[K_n] = \sum_{i=1}^{n} \frac{\alpha}{\alpha+i-1} \approx \alpha \log\left(1 + \frac{n}{\alpha}\right), \text{ for } n, \alpha >> 0.$$

The generalised method of moments consists of using the empirical measure of K_n to estimate $\hat{\alpha}$:

$$K_n = \hat{\alpha} \log\left(1 + \frac{n}{\hat{\alpha}}\right). \tag{2.8}$$

This estimation is not biased only with continuous base measures, for which draws are distinct with probability 1, while with atomic base distributions it is possible to draw the same atom multiple times, hence the empirical estimation of K_n must be corrected accordingly. These problems are carefully addressed in [28] for the Pitman-Yor process, for which the Dirichlet process is a special case and the distance-dependent CRP was shown here to benefit from the same parameter estimation techniques. [28] reviews 4 possible options to estimate the parameters, and we mention here and use in this work only the best performing one. There exist approximations for K_n, the number of unique tables after n customers have entered the restaurant, and H_{mn}, the number of tables with m customers after n customers have entered the restaurant, and these can be used for an empirical estimate of the parameters. In particular:

$$H_{1N} = \frac{\Gamma(1+\hat{\alpha})N^{\hat{d}}}{\Gamma(\hat{d}+\hat{\alpha})} \qquad K_N = \frac{H_{1N} - \hat{\alpha}}{\hat{d}}. \tag{2.9}$$

As anticipated, estimating K_n and H_{1N} requires particular attention with discrete base

measures, as a new draw from the base distribution could well be equal to previous samples, or, metaphorically, a new client at the restaurant could be assigned to a new table, but that table could be already occupied. Using a generalisation of the **birthday problem**, [28] obtain a correction to estimate $\hat{K}_n$ and $\hat{H}_{1n}$ given the observed $\tilde{K}_n$ and $\tilde{H}_{1n}$:

$$\hat{K}_n = \log\left(1 - \frac{\tilde{K}_n}{|\mathcal{V}|}\right)\log^{-1}\left(\frac{|\mathcal{V}|-1}{|\mathcal{V}|}\right) \qquad \hat{H}_{1n} = \tilde{H}_{1n}\left(\frac{|\mathcal{V}|}{|\mathcal{V}|-1}\right)^{\hat{K}_n - 1}. \tag{2.10}$$

In conclusion, all parameter estimations will use the correction Equation 2.10, PY and DDCRP will the use Equation 2.8 while PY will use Equation 2.9.

With respect to the estimation of the inhomogeneous Poisson process intensity, one first option was to draw some ideas from [31]. If we assume independence between days or weeks in terms of the point process, in the training phase we could consider each independent window as n independent realisations of the process. If the time axis is split into intervals of constant length δ and we define $l(t) = \lfloor \frac{t}{\delta} \rfloor \delta$, we can define an "aggregated" rate function constant on each interval $[l(t), l(t)+\delta)$:

$$\tilde{\lambda}(t) = \frac{1}{\delta}\int_{l(t)}^{l(t)+\delta} \lambda(s)ds,$$

and an estimator of the rate function that, for $n \to \infty$, converges almost surely to $\tilde{\lambda}(t)$ can be written as follows:

$$\tilde{\lambda}_n(t) = \frac{1}{n\delta}\sum_{i=1}^{n} N_i(l(t), l(t)+\delta), \tag{2.11}$$

where $N_i(a, b)$ denotes the number of events falling in the interval $[a, b)$ in the i-th independent observed realisation of the counting process N. However, such assumption of independency seemed pretty strong and we opted for a simpler solution. From Equation 2.4 we can write that:

$$\mathbb{E}[N(t+s) - N(t)] = \int_{t}^{t+s} \lambda(l)dl.$$

If we divide the time axis into intervals $[t_n, t_{n+1})$ that include only one occurrence, and we assume constant rate within such intervals, we can derive:

$$\lambda_n(t) = \frac{1}{t_{n+1} - t_n}, t \in [t_n, t_{n+1}). \tag{2.12}$$

The clear downside of estimating the rate parameter in such way is that it would be very noisy. For this reason, we decided to implement a forgetting factors mean estimation

[22]:

$$\bar{\lambda}_n(\Lambda) = \left(\sum_{i=1}^{n} \Lambda^{n-i}\right)^{-1} \left(\sum_{i=1}^{n} \Lambda^{n-i}\lambda_i\right),$$

where $\Lambda \in [0,1]$ is the forgetting factor: if $\Lambda \to 0$ the mean is estimated using only one observation, while as $\Lambda \to 1$ the mean is calculated using all observations. The calculation of such quantity is suited to a big data context, as it can be calculated sequentially like:

$$\bar{\lambda}_n(\Lambda) = \frac{s_n(\Lambda)}{w_n(\Lambda)},$$

with $s_n(\Lambda) = \Lambda s_{n-q}(\Lambda) + \lambda_n$ and $w_n(\Lambda) = \Lambda w_{n-1}(\Lambda) + 1$. The parameter Λ was tuned empirically, and generally takes quite a high value close to 0.99.

2.1.6. p-values combination

In all cases, whether we use a Dirichlet process, a Pitman-Yor process or a distance-dependent Chinese restaurant process, by applying the techniques presented above we calculate two p-values for each realised connection: a p-value for the destination and a p-value for the source given the destination. Only in the case where we also consider arrival times, we have a third p-value for the Poisson process at destination level. Calculating the joint p-value using the same concept as in Equation 2.3 would be possible in priciple, but intractable. For this reason we combine the two (or three) p-values using Fisher's p-value combiner to obtain a score for each edge (x, y):

Proposition 2.1.5. (Fisher's p-value combiner)

$$s = -2\sum_{i=1}^{k} \log \mathbf{p}_i \sim \chi^2_{2k}$$

The p-value, given the score, is calculated as the $SF(s)$ [2]. After all links have been assigned a score, similarly to [12, 28], we combine m scores on each link using a minimum p-value combiner, to obtain a singe score for each edge:

$$\mathbf{p}_{(x,y)} = \min\{\mathbf{p}_1, \cdots, \mathbf{p}_m\}.$$

The minimum p-value, due to discreteness and some possible correlation, is only approximately distributed like $Beta(1, m)$, with CDF $F(x|1, m) = 1 - (1 - x)^m$ used to calculate a p-value given the score. The final step is to calculate a score for each source node, which in this study is done again by using a minimum p-value combiner. The result is an sequence of source nodes ordered by p-value, with the nodes with smallest p-values potentially anomalous.

[2] Survival function

2.1.7. Code

The code for this project is available, as a library, at the following link `https://github.com/a-mazzetto/anomaly-detection/tree/final_project`. In particular:

- Functions for the various stochastic processes are in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/processes`;
- Synthetic data generation functions are in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/data_generation`, while scripts for generating synthetic datasets are in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/scripts/data_generation`;
- Parameter estimation scripts are in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/parameter_estimation` and scripts in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/scripts/parameters_estimation`;
- p-value calculations are defined in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/pvalues` and tested in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/scripts/pvalues_scripts`;
- The main scripts to launch the first part of anomaly detection are in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/anomaly_detection`.

2.2. Second Part of Anomaly Detection

The anomaly detection procedure studied up to here outputs a list of source nodes ordered by anomaly score, together with the timestamp indicating when the most extreme p-value was observed, before combining all p-values with the minimum p-value combiner. The choice of such combiner was done to have a meaningful timestamp signalling the anomaly, to be used in the second part of anomaly detection. If we wanted to take a more global view, the full dynamic graph would typically be intractable. However, we could focus on the neighbourhood of the potentially anomalous nodes, before and up to the potential anomaly, by constructing dynamic sub-graphs. Such graphs can be assigned a probability based on a model of "normal" behaviour - in our case a dynamic graph variational auto-encoder, DynVAE, trained on sub-graphs thought to be anomaly-free - to allow for a second ranking of the nodes based on their unlikelihood given the model. Before developing our DynVAE model, we carried out some preliminary work to:

- Compare the performance of the most common graph convolution layers, GCN, GraphSAGE and GIN, on a graph classification task based on a synthetic dataset, to choose which one to adopt in our variational auto-encoder's encoder. Theoretically, we expect the GIN layer to be the better suited for our application, where

optimal discrimination based on graph topology is desirable. Theoretical details on the different types of convolutions can be found in Appendix C;

- Train a static graph variational auto-encoder on one of the classes from the previous point, and verify its capability to discriminate between the different classes in prediction, by calculating the likelihood of a graph given the model. The idea of this experiment was to prove that the proposed approach would work in a simple example, before moving to the dynamic case. For details on the variational auto-encoder used, please refer to Appendix D.

These preliminary steps will not be further discussed in this chapter, but the results will be presented in subsection 4.1.3. The section proceeds as follows: first we describe how the dynamic sub-graphs are constructed, before introducing the details of our DynVAE model.

2.2.1. Dynamic graphs construction

After the first part of anomaly detection, each of the $|\mathcal{V}|$ nodes in the network $\mathcal{G}$ is associated with an anomaly score s_i and time of potential anomaly t_i. In the second part of anomaly detection we wish to use this information, together with the marked point process that defines the evolution of the entire network, to build dynamic graphs comprising the history of the neighbourhood of each potentially anomalous node up to the time of anomaly detection. Let us assume that we want to construct the dynamic graph $\mathcal{G}_T^{(x_i,t_i)} = \left(\mathcal{V}_T^{(x_i,t_i)}, \mathcal{E}_1^{(x_i,t_i)}, \cdots, \mathcal{E}_T^{(x_i,t_i)}\right)$ for node $x_i \in \mathcal{V}$ with potential anomaly at time t_i, time aggregation $\delta \in \mathbb{R}^+$ and $T \in \mathbb{N}$ time steps. We define the node set as:

$$\mathcal{V}_T^{(x_i,t_i)} = \{x \in \mathcal{V} | x \in \mathcal{N}^{out}_{k,[t_i+(0.5-T)\delta, t_i+0.5\delta]}(x_i)\},$$

where $\mathcal{N}^{out}_{k,[t_0,t_1]}$ is the neighbourhood of x_i defined as the nodes that can be reached from x_i with a out-directed k-steps path in the time interval $[t_i+(0.5-T)\delta, t_i+0.5\delta]$. We look at the out-directed neighbourhood because for a node to be able to spread anomaly, it must be reached by the potentially anomalous node x_i first. The edge set changes with time and is defined as:

$$\mathcal{E}_j^{(x_i,t_i)} = \{(x_i, x_j) \in \mathcal{V}_T^{(x_i,t_i)} \times \mathcal{V}_T^{(x_i,t_i)} : N_{[t_i+(0.5-(T+1-j))\delta, t_i+(0.5-(T-j))\delta]}(x_i, x_j) > 0\},$$

with $j = 1, \cdots, T$ and $N_{[t_0,t_1]}$ a counting process in the defined time interval $[t_0, t_1]$. In this work we fixed $T = 4$ and experimented with $\delta \in \{5, 15, 30\}$ minutes, hence considering a total time interval between 20 minutes and 2 hours. In general, the choice of the aggregation interval δ must be coherent to the data, to avoid consistently empty or fully connected sub-graphs. We put constraints on the dimension of $\left|\mathcal{V}_T^{(x_i,t_i)}\right| > 10$, to exclude small sub-graphs, and on $0 < \left|\mathcal{E}_j^{(x_i,t_i)}\right| < \left|\mathcal{V}_T^{(x_i,t_i)}\right|^2$ to exclude cases without links or fully connected that create issues in the calculation of losses and metrics. Dynamic graphs for model training are created in a similar fashion, for all nodes active in the

training period. We wish to use the anomaly score s_i together with in-degree and out-degree as node features.

2.2.2. DynVAE Model

Our dynamic graph variational auto-encoder, DynVAE in short, is a slight extension of the model presented in [10], to support directed graphs and allow for extra flexibility by means of a Gaussian mixture prior. The authors extend the graph variational auto-encoders [15] to dynamic graphs by allowing for a prior distribution that evolves with time, and the evolution of which is regulated by a graph recurrent neural network. Please see Appendix D for details on the (static) variational auto-encoder for graphs. Let us consider a dynamic graph $\mathcal{G} = \{\mathcal{G}_t\}_{t=1}^{T}$ with adjacency matrices $\{A_t \in \mathbb{R}^{n_t} \times \mathbb{R}^{n_t}\}_{t=1}^{T}$ and node attribute matrices $\{X_t \in \mathbb{R}^{n_t} \times \mathbb{R}^{d}\}_{t=1}^{T}$ with a potentially different number of nodes at each time-step $n_t = |\mathcal{V}_t|$. The prior distribution for the embedding matrix $Z^{(t)}$ is assumed to be:

$$p\left(Z^{(t)}\right) = \prod_{i=1}^{n_t} p\left(z_i^{(t)}\right) \quad \text{with} \quad p\left(z_i^{(t)}\right) = \mathcal{N}\left(z_i^{(t)} \middle| \mu_{i,prior}^{(t)}, diag\left(\left(\sigma_{i,prior}^{(t)}\right)^2\right)\right).$$

The key is that the parameters of the prior distribution vary with time, and the matrix of stacked means and standard deviations depends on a state variable by means of a highly flexible function, potentially a neural network:

$$\{\mu_{prior}^{(t)}, \sigma_{prior}^{(t)}\} = \varphi^{(prior)}(h_{t-1}).$$

The recurrent hidden state is updated with a graph recurrent neural network, where, for flexibility, φ^x and φ^z functions are introduced:

$$h_t = f\left(A^{(t)}, \varphi^x(X^{(t)}), \varphi^z(Z^{(t)}), h_{t-1}\right). \tag{2.13}$$

The remainder is very similar to what seen in Appendix D for the variational auto-encoder, but replicated for each time-stamp. The encoder is:

$$q_\phi(Z^{(t)}|X^{(t)}, A^{(t)}, h_{t-1}) = \prod_{i=1}^{n} q_\phi(z_i^{(t)}|X^{(t)}, A^{(t)}, h_{t-1}),$$

with:

$$q_\phi(z_i^{(t)}|X^{(t)}, A^{(t)}, h_{t-1}) = \mathcal{N}(z_i^{(t)}|\mu_i^{(t)}, diag((\sigma_i^{(t)})^2)).$$

The matrix of mean vectors $\mu_i^{(t)}$ as rows is calculated as:

$$\mu^{(t)} = GNN_\mu\left(A^{(t)}, CONCAT(\varphi^x(X^{(t)}), h_{t-1}))\right),$$

and similarly:

$$\log \sigma^{(t)} = GNN_\sigma \left(A^{(t)}, CONCAT(\varphi^x(X^{(t)}), h_{t-1})) \right),$$

where GNN_μ and GNN_σ are graph neural networks. In order to support directed graphs, our approach has been to allow for two latent spaces, one with the projections of source nodes, the other with the projections of destination nodes; in turn, the procedure described up to now is duplicated for source and destination embeddings denoted by Z^s and Z^d respectively. The generative model uses drawn vectors z_i^s and z_i^d, depending on whether the node is source or destination, to reconstruct the adjacency matrix:

$$p_\theta(A|Z^s, Z^d) = \prod_{i=1}^{n} \prod_{j=1}^{n} p_\theta(A_{ij}|z_i^s, z_j^d) \text{ with } p_\theta(A_{ij} = 1|z_i^s, z_j^d) = SIGMOID(z_i^{sT} \cdot z_i^d),$$

where the apex (t) was dropped for simplicity of notation. The assumption is that $A_{ij}|Z^s, Z^d \overset{iid}{\sim} Bernoulli(p_\theta(A_{ij} = 1|z_i^s, z_j^d))$. Model parameters are optimised by minimising the Evidence Lower Bound, with binary cross-entropy loss and KL-divergence from a prior distribution as described before, similarly to [10]. The second contribution to DynVAE is the introduction of a Gaussian mixture prior, with m modes, to add extra flexibility to the model as necessary. The model was developed in PyTorch, and featured tunable GNN hidden dimension, tunable latent space dimension, multi-layer perceptrons with a tunable number and dimension of hidden layers for φ^x and φ^z, graph recurrent neural network for the prior state with a tunable number of graph convolution layers and a tunable number of prior modes. Following the reasoning and testing of graph convolution layers, we adopt GIN convolution layers in our model, with a tunable number of message passing layers. Figure 2.3 is a visualisation of the implemented network.

Once that the model is trained, we wish to calculate the **likelihood** of an unseen graph of type $\mathcal{G}_T^{(x_i,t_i)}$, with notation as defined in the previous section, with adjacency matrices $A_1^{(x_i,t_i)}, \cdots, A_T^{(x_i,t_i)}$. The probability, where for simplicity we drop the apex (x_i, t_i), is:

$$p_\theta\left(A_1, \cdots, A_T\right) = \prod_{t=1}^{T} p_\theta\left(A_t|\{A_\tau : \tau \leq t\}\right) =$$

$$\prod_{t=1}^{T} \int_{Z^s, Z^d} p_\theta\left(Z^s, Z^d|\{A_\tau : \tau \leq t\}\right) p_\theta\left(A_t|Z^s, Z^d, \{A_\tau : \tau \leq t\}\right) dZ^s dZ^d. \quad (2.14)$$

Such integral is approximated with Monte Carlo methods, with $N = 1000$ samples in our case. We end up with a matrix of Bernoulli probabilities for each time step t that is:

$$p_\theta(A_t|\{A_\tau : \tau \leq t\}) = \prod_{i=1}^{|\mathcal{V}_T|} \prod_{j=1}^{|\mathcal{V}_T|} p((A_t)_{ij} = 1).$$

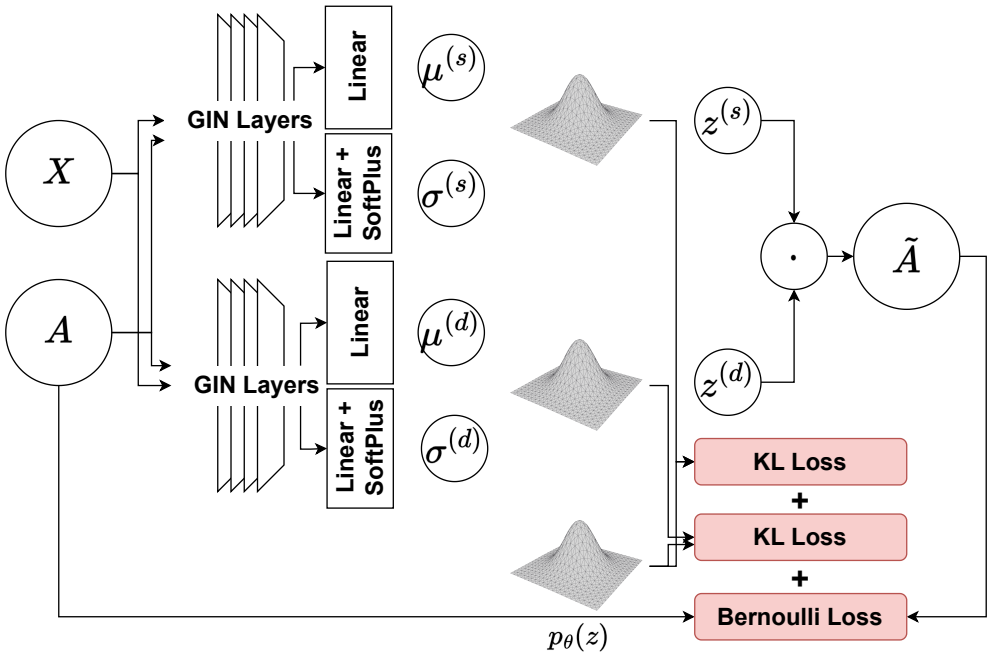

Figure 2.3.: Graph Variational Auto-encoder Schema

Given that each entry of the adjacency matrix is assumed to be Bernoulli identically and independently distributed, we can calculate the likelihood of an observed dynamic graph - given the model - as:

$$\log \mathcal{L}_t = \sum_{i=1}^{n} \sum_{j=1}^{n} \left[p_\theta(A_t) \cdot A_t + (1 - p_\theta(A_t)) \cdot (1 - A_t) \right]_{ij}, \tag{2.15}$$

where all operations are element-wise. In order to make likelihoods comparable across different matrix dimensions, we work with normalised logarithmic likelihood, with respect to a random model which gives probability 0.5 to all entries of the adjacency matrix:

$$\log \mathcal{L}_t^* = \frac{\log \mathcal{L}_t}{n^2 \log 0.5}.$$

This was the first approach proposed to rank potential anomalies according to the second part of anomaly detection. One other possibility is to work with **Bernoulli p-values** or mid p-values. For a Bernoulli random variable $X \sim Bernoulli(p)$ the right p-value of a realized value x is:

$$\mathbf{p}_R(X = x) = \begin{cases} 1.0 \text{ if } p(X = x) \geq 0.5 \\ p(X = x) \text{ if } p(X = x) < 0.5, \end{cases} \tag{2.16}$$

and, similarly, the left p-value is:

$$\mathbf{p}_L(X = x) = \begin{cases} 1.0 - p(X = x) \text{ if } p(X = x) \geq 0.5 \\ 0.0 \text{ if } p(X = x) < 0.5. \end{cases} \tag{2.17}$$

As already used before, the p-value is expressed as $\mathbf{p} = 0.5(\mathbf{p}_L + \mathbf{p}_R)$. Once the p-value has been calculated for each entry of the adjacency matrix, we combine all entries into a single p-value using Fisher's combination method 2.1.5, which naturally adjusts for different matrix dimensions without need of further normalizations. We will use both methods to calculate the likelihood or p-value of a dynamic graph and compare their performance.

2.2.3. Final Anomaly Ranking

The first part of anomaly detection outputs a p-value, while the second part of anomaly detection might output a normalised log-likelihood or a p-value. When both parts output a p-value, the combination can once again be done with Fisher's p-value combiner 2.1.5. When the output of the two parts is heterogeneous, we treat the two rankings as independent, equally weighted rankings and rigorously combine them into a single ranking using Schulze's method [29], a ranking method used in voting systems.

2.2.4. Code

The code for this project is available, as a library, at the following link `https://github.com/a-mazzetto/anomaly-detection/tree/final_project`. In particular:

- Deep learning models in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/gnn`, application in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/scripts/gnn`;
- Custom PyTorch datasets in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/gnn_data_generation`;
- Main scripts for part two of anomaly detection in `https://github.com/a-mazzetto/anomaly-detection/tree/final_project/scripts/part2`.

3. Data

Following the data generating process described in chapter 2, we have created synthetic data using both a PY hierarchical process or a DDCR hierarchical process, and added a simulated anomaly with lateral movement. The techniques have also been tested on the Los Alamos National Laboratory Windows authentication dataset [36], openly available at the following link.

3.1. Synthetic Data

In this section we are concerned with the creation of a marked point process $(\mathcal{T}, \mathcal{X}, \mathcal{Y}) = \{(T_i, X_i, Y_i)\}_{i\geq 1}$ where $\mathcal{T} \sim \mathcal{PP}(\lambda(t))$ and the random variables X_i and X_j follow the hierarchical model 2.2.

Starting from the Poisson point process, the choice of intensity parameter has been done to mimic the intensity and periodicity of the LANL dataset, with 12000 nodes and daily activity that ranges between 0.8 and 1.4 million connections every 4 hours. Based on this, we have identified:

$$\lambda_{min}(|\mathcal{V}|) = 4.6e^{-3}|\mathcal{V}| \quad \lambda_{max}(|\mathcal{V}|) = 8.1e^{-3}|\mathcal{V}|,$$

and postulated the following inhomogeneity for the rate parameter of the inhomogeneous Poisson process:

$$\lambda(t) = \lambda_{mean}(|\mathcal{V}|) + \lambda_{delta}(|\mathcal{V}|)\sin\left(\frac{2\pi}{T_0}t\right), \tag{3.18}$$

where $\lambda_{mean}(\cdot) = 0.5(\lambda_{min}(\cdot) + \lambda_{max}(\cdot))$, $\lambda_{delta}(\cdot) = 0.5(\lambda_{min}(\cdot) - \lambda_{max}(\cdot))$ and T_0 is a periodicity parameter. Such an assumption, makes it possible to define the maximum rate:

$$\lambda^* = \lambda_{mean}(|\mathcal{V}|) + \lambda_{delta}(|\mathcal{V}|),$$

and generate data for such inhomogeneous Poisson process by means of thinning (see, for example, [31]).

Definition 3.1.1 (Thinning algorithm for non-stationary Poisson process with $\lambda(t)$ upper bounded by λ^*)**. Requires**: intensity $\lambda(t)$, upper bounded by λ^*, and maximum time T. **Ensures**: realisation $t_1, \cdots, t_{N(t)}$ of the process.

1. $t = 0$, $N = 0$;
2. Generate an inter-arrival time $\delta \sim Exp(\lambda^*)$ and set $t = t + \delta$. Stop if $t > T$;
3. Generate $u \sim Unif(0, 1)$;

4. If $u \leq \lambda(t)/\lambda^*$ set $N = N + 1$ and $t_N = t$;

5. Restart from 2.

Once that the time sequence has been generated, we can proceed with the destination sequence generation according to 2.2. If the generating process is a Pitman-Yor process, it is possible to generate the data using the Chinese restaurant formulation:

Definition 3.1.2 (Generation of sequence of data according to the Pitman-Yor process)**. Requires**: number of elements $|\mathcal{V}|$ from which the uniform prior distribution follows, with $p_0 = 1/|\mathcal{V}|$ **Ensures**: realisation $\{x_1, \cdots, x_N\}$ of the process.

1. $n = 1$, draw a sample x_n from the uniform base measure, set $\mathbf{x}_n = \{x_n\}$ and let $\mathbf{x}_n^* = \{x_n\}$, set of unique elements of $\mathbf{x}_n$, with $K_n = |\mathbf{x}_n^*|$;

2. Roll a dice with faces $\{-1\} \cup \mathbf{x}_n^*$ and probabilities

$$\left\{ \frac{\alpha + dK_n}{\alpha + n} \right\} \cup \left\{ \frac{N_i - d}{\alpha + n} \right\}_{i=1}^{K_n};$$

3. If the outcome is -1, draw a new sample x_{n+1} from the base measure, otherwise set $x_{n+1} = x_j^*$ where j is the element drawn from $\mathbf{x}_n^*$;

4. Update the sequence $\mathbf{x}_{n+1} = \{x_1, \cdots, x_{n+1}\}$, update the set $\mathbf{x}_{n+1}^*$ of unique values and its multiplicity K_{n+1}. $n = n + 1$;

5. Repeat from 2.

The procedure is analogous for the distance-dependent Chinese restaurant process, but we draw a new sample from the base measure with probability $\alpha/(\alpha + n)$, or an existing atom x_i with probability $\alpha/(\alpha+n) \sum_{j=1}^{N_i} f(d_j) / \sum_{j=1}^{n} f(d_j)$, where d_i is the time distance from current time and N_i is the number of occurrences of x_i. Once either process has been generated for the destination nodes, independent processes for each destination node can be instantiated to generate source nodes. The generated datasets can be visually inspected by plotting table assignment against customer number. For example, by comparing 3.1 and 3.2 we can see the effect of the discount parameter, which for high values gives a much faster appearance of new nodes, discounting previous observations. Comparing to 3.3 instead, it is possible to see the effect of DDCRP, with certain nodes appearing only once or a few times when the decay parameter is quite high.

3.2. Synthetic Lateral Movement

So far we described the data generating process and simulated it. In order to test our anomaly detection procedure, however, we also need to simulate an intrusion. One characteristic that our anomaly should have is a lateral movement pattern [7, 8, 4]. The problem of simulating such intrusions is of ordinary interest for red teams, and

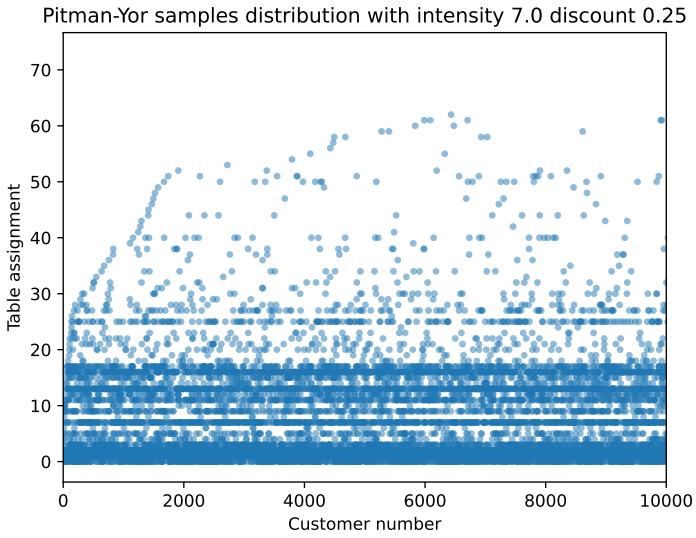

Figure 3.1.: PY generating process with intensity 7.0 and discount 0.25

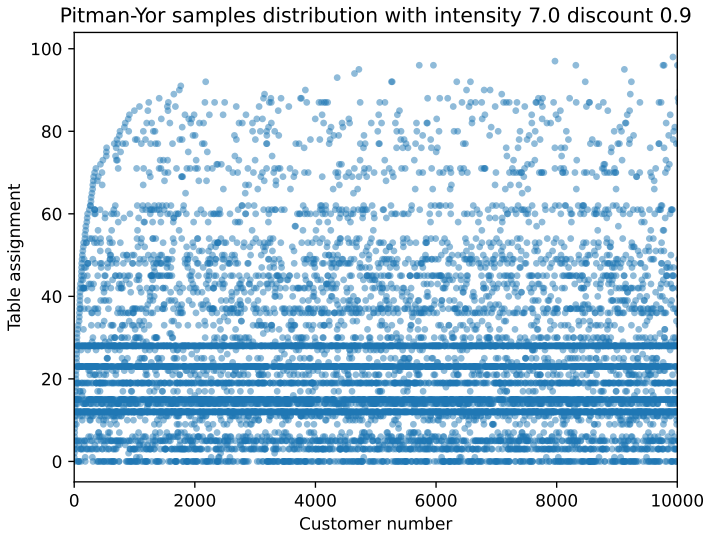

Figure 3.2.: PY generating process with intensity 7.0 and discount 0.9

the literature and available simulations in the field is vast, see for example [33]. In the context of our study, however, we opted for simpler simulations of intrusion. The intrusion simulation requires two specifications:

1. Choice of start and target machines of the lateral movement;
2. Lateral movement pattern.

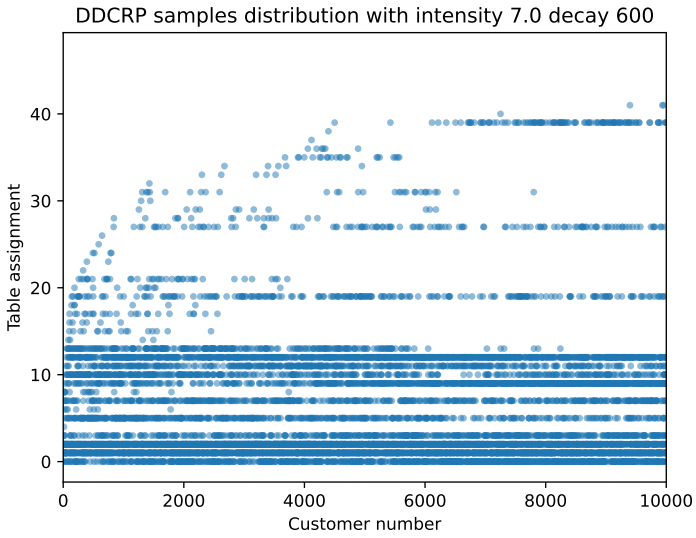

Figure 3.3.: DDCRP generating process with intensity 7.0 and decay constant $600s$

The simulation starts with an observation period of network activity, after which a static graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is created. After such period of observation, all nodes are assigned an importance metric such as PageRank. The selection of the lateral movement endpoints is done as follows: the starting point is reasonably a liable, low activity computer that is randomly selected between the 75% least "important" computers based on the PageRank metric, while the target computer could be either another low importance computer or one amongst the 25% of most important computers. The rationale behind the choice of type of target is whether we would like to access a high traffic computer or a remote computer. Let us call the source and target computers x_S and x_T respectively. We implemented two possibilities to move between the two. The first is based on random walks on the observation graph $\mathcal{G}$: M random walks of length N are set out from x_S reaching $x_{1,j}$ with $j = 1, \cdots, M$, then other M random walks of length N are set out from each $x_{1,j}$ to reach $x_{2,j}$ with $j = 1, \cdots, M^2$ and so on, until the target computer is reached. In alternative, x_T is reached from x_S following the longest path in $\mathcal{G}$ and attempting each connection M times. We decided to use information from the observation graph $\mathcal{G}$ is to mimic a "clever" intrusion, where information learnt during the lateral movement would be progressively used to improve the subtlety of the lateral movement.

3.3. LANL authentication dataset

The *Comprehensive, Multi-Source Cyber-Security Events* dataset consists of 58 consecutive days of Windows-based authentication events, with $1s$ time resolution, and is openly available at the following link [36]. Failed authentications are recorded only for users who had at least one successful authentication within the dataset. `NA`s are represented

by ?. A one-line sample of the dataset is shown below, where the following information can be found: time, source user@domain, destination user@domain, source computer, destination computer, authentication type (such as NTLM, which is an older challenge-response authentication protocol, or Kerberos, that uses third-party verification and stronger encryption), logon type, authentication orientation and success/failure.

```
1,ANONYMOUS LOGON@C586,ANONYMOUS
LOGON@C586,C1250,C586,NTLM,Network,LogOn,Success
```

For our analysis we will be concerned with time, source and destination information, exclusively for `LogOn` activity. Figures 3.4 and 3.5 show 4-hourly and weekly activity in the dataset, and highlight the expected periodicities.

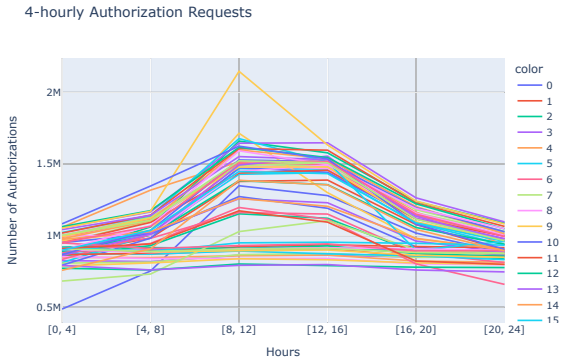

Figure 3.4.: Number of 4-hourly authorisation requests in the LANL dataset

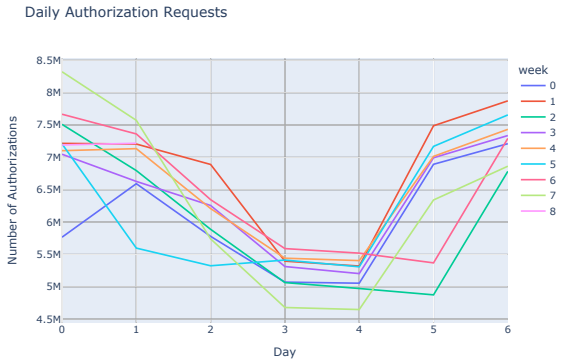

Figure 3.5.: Number of weekly authorisation requests in the LANL dataset

4. Results

In this chapter we discuss the outcomes of applying the techniques discussed so far to both synthetic and real data. In particular, we calculate scores for the first part of anomaly detection using the model approaches previously discussed: Pitman-Yor and Dirichlet hierarchical processes, Poisson marked point process with Pitman-Yor hierarchy in source and destination nodes, streaming version of the Pitman-Yor process and distance-dependent Chinese restaurant process. Follows a second part of anomaly detection using a dynamic graph variational auto-encoder model to express either the likelihood or p-value of opportunely built potentially anomalous dynamic graphs, using the scores from part one as features.

All calculations related to the synthetic dataset, except deep learning model training, were run on a DELL XPS 12 7390 laptop with 8GB RAM and processor Intel i7-10710U. Calculations for the LANL dataset were performed on a Linux cluster, with two Intel Xeon X5670 CPUs, 12 cores, 24 threads and 192GB of memory. Deep learning models training was done on a GPU cluster, with eight nVidia GTX 2080 Ti GPU cards, 1.5 Tb of memory and 22 TB of local data storage. The GPUs each have 4352 cores and 68 ray-tracing cores as well as 12 GB of memory.

4.1. Simulation

4.1.1. Bayesian processes parameter estimation

The first evaluation that can be done on synthetic data is the goodness of parameter estimation. We generated 10000 Pitman-Yor sequences of length 10000 with $\alpha = 7.0$ and $d = 0.25$, and estimated the parameters using the techniques from subsection 2.1.5 as shown in Figure 4.1. This is a replication of what first shown in [28], and we confirm the goodness of estimation also on our datasets. We also simulated 10000 distance-dependent CRP sequences of length 1000 with $\alpha = 7.0$, and performed parameter estimation as shown in Figure 4.2. Finally, Figure 4.3, shows parameter estimation for the inhomogeneous Poisson process, as explained in subsection 2.1.5, with $\Lambda = 0.99$.

4.1.2. First Part of Anomaly Detection

48 hours of activity within a small enterprise network with $|\mathcal{V}| = 100$ have been simulated with periodicity $T_0 = 12h$ in Equation 3.18 for the timestamp, and with two cases for node sequences:

- PY hierarchical process with $\alpha = 7.0$ and $d = 0.25$ for the destination process and with α sampled from $\{2.0, 7.0, 12.0\}$ with probabilities $\{0.6, 0.3, 0.1\}$, d sampled

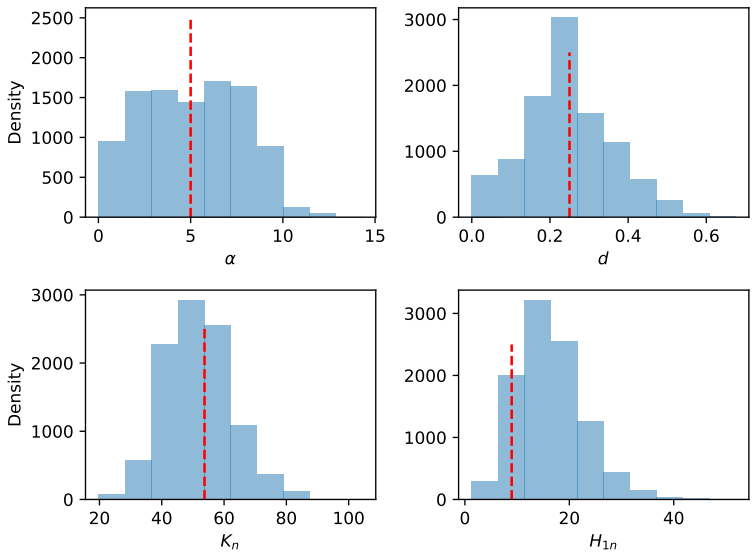

Figure 4.1.: Parameter estimation for 10000 Pitman-Yor sequences of length 10000 with $\alpha = 7.0$ and $d = 0.25$. Red dotted lines: true values

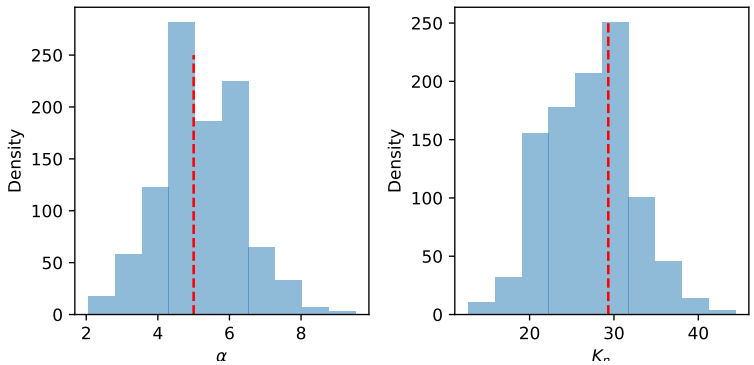

Figure 4.2.: Parameter estimation for 10000 distance-dependent CRP sequences of length 1000 with $\alpha = 7.0$. Red dotted lines: true values

from $Beta(2.0, 5.0)$ for the source processes.

- Hierarchical process destination sequence as before and source sequences following a DDCRP with intensity α sampled as before, and decay constant $\beta = 12h$.

A random walk anomaly, as explained in section 3.2, has been inserted after 36 hours, with walk length 10 and 2 walks set out at each iteration until the target node is reached.

The anomaly detection procedure has been carried out with all model options, with training period $[0, 24)$ hours and deployment period $[24, 48]$ hours. During the training period Bayesian non-parametric process parameters α and d are estimated and all sequences are updated but no p-value is calculated. The five options are as follows:

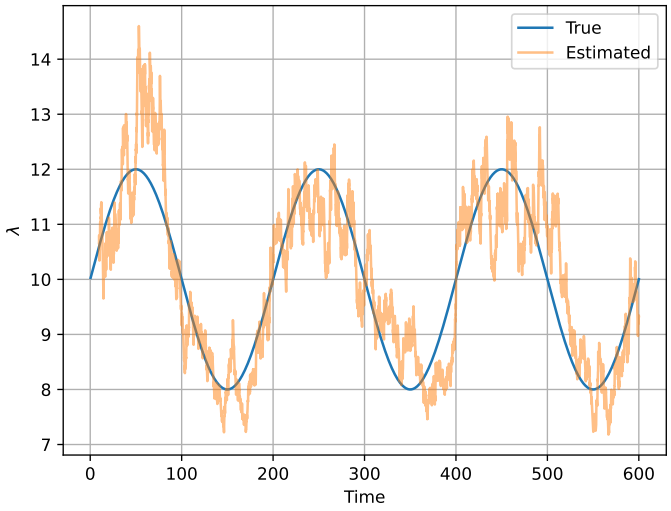

Figure 4.3.: Rate parameter estimation for an inhomogeneous Poisson process

1. p-values calculated considering a hierarchical Pitman-Yor model, like in [28];
2. p-values calculated considering a hierarchical Dirichlet model, similarly to [12] but with a process also for destinations;
3. p-values calculated with a hierarchical distance-dependent Chinese Restaurant Process, with decay constant 12 hours, like in the generating process (although, in the generating process the DDCRP is only for sources);
4. p-value calculated with a hierarchical streaming Pitman-Yor, with streaming window of length 12 hours;
5. p-values calculated using Poisson point process estimation for the destination arrivals, with rate estimated with a forgetting factors mean with $\Lambda = 0.99$, and a hierarchical Pitman-Yor like in 1. for the node sequences.

The anomaly ranking presented by each of the methodologies for each of the two datasets is analysed by quoting the following numbers:

- Number of truly anomalous nodes ranked in the top 10 or top 20;
- Overall improvement of anomaly ranking with respect to the dataset without anomaly;
- Overall relative improvement of anomaly ranking with respect to one of the methodologies taken as reference.

Dataset 1	PY	DP	DDCRP	Stream PY	Poisson and PY
N. of anomalies in top 10	6	5	6	9	7
N. of anomalies in to 20	10	8	11	13	11

Table 4.1.: Number of true anomalous source nodes in top 10 and top 20 for dataset 1 (generated with hierarchical PY assumptions)

Dataset 1	PY	DP	DDCRP	Stream PY	Poisson and PY
Wilcoxon p vs. no anomaly	0.67	0.88	0.94	0.86	0.89
Wilcoxon p vs. reference	*	0.95	0.21	0.10	0.79

Table 4.2.: Wilcoxon test p-value calculated for ranking of all truly anomalous source nodes, against the same methodology on a dataset without anomaly and against other methods, for dataset 1 (generated with hierarchical PY assumptions)

The first point is self-explanatory: as the anomaly is simulated, we quote the number of truly anomalous nodes which fall at the top of the ranking, and which would likely be the anomalies addressed first by the security team. As this is not the full picture and there is extra information we can extract from this simulated setting, we look at how the ranking changed for all truly anomalous source nodes, with respect to the same procedure run on the dataset without anomaly, and with respect to other methodologies. Let us assume that we want to compare the first method ranking $\{(x_i, r_i)\}_{i=1}^k$, with $r_i \in \{1, 2, \cdots, |\mathcal{V}|\}$ and where the smaller r_i the higher the anomaly level, to the ranking provided by a second method $\{(x_i, r'_i)\}_{i=1}^k$, believed to improve upon the first. The null hypothesis is that the rankings for method two are equal or higher (i.e., method two is equal or worse), or equivalently that the distribution of differences $r_i - r'_i$ is stochastically equal or shifted to the left with respect to a distribution symmetric with respect to zero. The alternative hypothesis is that the distribution for the second method is shifted to the right. This hypothesis can be tested rigorously using the Wilcoxon signed-rank test.

Regarding the application to the first dataset, Table 4.1 shows that streaming Pitman-Yor and Poisson marked point process are amongst the best models at placing anomalous nodes on top. The Wilcoxon test in Table 4.2 shows that we cannot reject the null hypothesis with respect to the same procedure applied on the dataset without anomaly, in all cases. This is related to the fact that while some nodes gain positions in the anomaly ranking, others lose position, a phenomenon that is particularly emphasised in a small dataset with only 100 nodes of which 36 are truly anomalous. As one would expect, the methodology that, despite timidly, shows the most evidence against the null hypothesis is the PY model, which is the same as the generating process. The second row of Table 4.2 shows that the only methodologies that resemble the overall performance of PY are DDCRP and Streaming PY. The latter, overall, would seem the best performing model in this scenario.

On the application to the second dataset, Table 4.3 shows that in this case the situation

Dataset 2	PY	DP	DDCRP	Stream PY	Poisson and PY
N. of anomalies in top	5	5	5	6	4
N. of anomalies in to	11	11	11	9	9

Table 4.3.: Number of true anomalous source nodes in top 10 and top 20 for dataset 2 (generated with hierarchical PY and DDCR processes)

Dataset 2	PY	DP	DDCRP	Stream PY	Poisson and PY
Wilcoxon p vs. no anomaly	0.94	0.88	0.94	0.80	0.76
Wilcoxon p vs. reference	0.08	0.29	*	0.10	0.18

Table 4.4.: Wilcoxon test p-value calculated for ranking of all truly anomalous source nodes, against the same methodology on a dataset without anomaly and against other methods, for dataset 2 (generated with hierarchical PY and DDCR processes)

is less clear: PY, DP and DDCRP perform equally and probably best. The Wilcoxon test in Table 4.4 shows that we cannot reject the null hypothesis with respect to the same procedure applied on the dataset without anomaly also in this case. The other observation is that none of the methodologies matches exactly the generative process, and in fact none stands out from the test. Using information from the second row of Table 4.4, the PY model is the closest to challenging DDCRP, and together they are probably two of the best performing models overall (and, perhaps unsurprisingly, both are part of the data generating process).

4.1.3. How powerful are GNNs?

The title of this subsection is a citation from [41]. This is the description of a set of experiments that were performed on the synthetic dataset to choose what **graph convolution layer** type to use for DynVAE, and to gain confidence over the potential of the proposed approach for the second part of anomaly detection, that is to **discriminate between different graphs** by using a variational auto-encoder reference model. We created a graph dataset called `ThreeGraphFamiliesDataset` which is constituted by 15 minute aggregations of dynamic graphs generated by three different processes, for a total of 1500 graphs, 500 per type. Each group of data is generated by 5 instances of a 24 hours long sequence of links following a Pitman-Yor hierarchical process with the same parameters $\alpha = 7.0$ and $d = 0.25$ for the destination process, but different parameters for the source processes:

- α drawn with probabilities $\{0.6, 0.3, 0.1\}$ from $\{2.0, 7.0, 12.0\}$ and $d = 0.25$ for the first group;
- $\alpha = 0.7$ and $d = 0.25$ indistinctively for the second group;
- $\alpha = 0.7$ and $d = 0.75$ indistinctively for the third group.

There exist two versions of the dataset, one without node covariates and one with in-degree and out-degree as node covariates.

To choose what **graph convolution layer** type to use, we built a classification model in PyTorch with: 3 graph convolution layers (either GraphSAGE, GCN or GIN) with input dimension either 1 (with no node features) or 2 (with node features), hidden dimension 64 and training dropout 10%, followed by a global mean pooling layer, a 10% dropout layer and finally a linear layer to adjust the output - interpretable as logits - to the classification dimension 3, equal to the number of classes. The model parameters were optimized with an Adam algorithm with learning rate $5e^{-4}$ and weight decay $5e^{-4}$, cross entropy loss, and maximum 1000 epochs with patience 100. Figures 4.5 and 4.5 confirm that, both with or without node features, GIN convolution is the best or amongst the best convolution layers. Following this observation, we decide to use GIN layers in our GNNs unless stated otherwise.

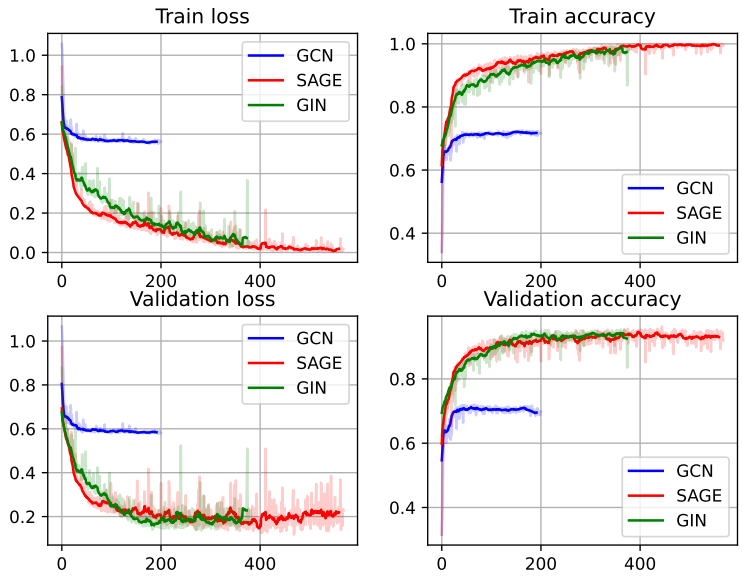

Figure 4.4.: Training and test results, following a (70%, 30%) split, for a graph classification task with node attributes

Secondly, we verified if a graph variational auto-encoder would be capable of distinguishing between the same three graph categories, when trained on the first category only. We created a dataset with 5000 graphs called `OneGraphFamilyDataset`, generated with the same process as explained for group one, and trained a (static) graph variational auto-encoder equivalent to the DynVAE model in subsection 2.2.2 with $T = 1$, hence only one time step and no GRU layers. We scan the following hyper-parameters: latent dimension 16 or 32, with hidden dimension 32 or 64 respectively, prior with 1 or 3 modes and binary cross-entropy positive class weights 1, 15 or proportional to the number of positive entries in the adjacency matrix (Table 4.5). In fact, the graph is sparse, and the binary cross-entropy loss suffers of the issues related to class unbalance.

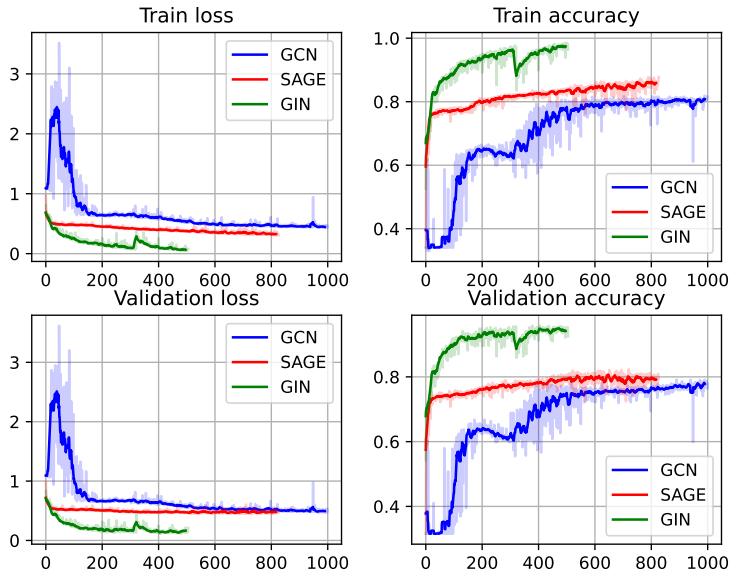

Figure 4.5.: Training and test results, following a (70%, 30%) split, for a graph classification task without node attributes

As we deal with relatively small graphs, we decided to fix the number of GIN convolution layers to 3, so to consider the 3-hop neighbourhood of any node. The variational auto-encoder was trained for up to 2000 epochs with patience 200, using an Adam optimisation algorithm with learning rate $0.5e^{-4}$ and weight decay $0.5e^{-4}$. The results have been evaluated in terms of area under the ROC curve, likelihood 2.15 and confusion matrix between predictive probabilities of the adjacency matrix entries - obtained with a 10000 samples Monte Carlo simulation 2.14 - and the true adjacency matrix. The predictive probabilities were threshold at 0.5 to calculate the confusion matrix. Figure 4.6 shows, for one example, the true adjacency matrix together with the non-threshold model predictive probability. We can notice clearer "vertical" patterns, linked to the fact that the data is generated with a relatively low intensity for the destination process for $\mathbb{P}_{\mathcal{Y}}$, hence showing quite high recurrence in some of the destinations.

AUC and logarithmic p-value were calculated also for graphs of type II and III, not used to train the model, to verify the discriminatory capacity of the model. Recall that types I, II and III correspond to groups I, II and III used for the previous example of classification. The results of the parameter scan are reported in Table 4.6: higher latent and hidden dimensions and proportional positive class weight give the best results in terms of AUC and TP rate, while the difference between 1 and 3 prior modes is almost negligible. Figure 4.7 shows the log-likelihood and p-value for the three classes of graphs in `ThreeGraphFamiliesDataset`, of which the first is the one used to train the model; the model is able to discriminate between the three classes, both in terms of log-likelihood and p-value, and this gives confidence on the possibility to use such approach to find anomalous sub-graphs.

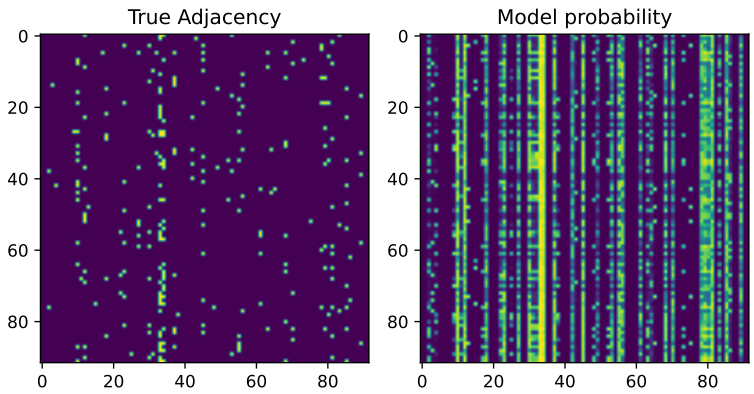

Figure 4.6.: One sample from dataset `OneGraphFamilyDataset`: true adjacency matrix together with model predictive probabilities obtained with a 10000 samples Monte Carlo simulation

Model	Latent/Hidden dimension	Prior modes	Positive class weight
a	16/32	1	1
b	16/32	1	15
c	16/32	1	$\propto$ positive entries
d	32/64	1	$\propto$ positive entries
e	32/64	3	$\propto$ positive entries

Table 4.5.: Table of scanned hyperparameters

There are two main takeaways from this transitory chapter: GIN graph convolution layers, as argued in [41], are powerful at discriminating graph topologies and are the best suited for our application, and the idea of using the likelihood or p-value of a graph, given a trained variational auto-encoder model, to discriminate between different patterns or topologies produced promising results in a simple classification problem.

4.1.4. Second Part of Anomaly Detection

The DynVAE model used in the second part of anomaly detection has latent dimension 32, hidden dimension 64, 3 GIN layers in the encoders, 5 GRU layers between prior updates, and 1 dense layer for φ^x and φ^z in Equation 2.13. Empirical tests showed that using positive class weight proportional to the number of positive entries in the adjacency matrix performed well also in this case, and similarly a mixture prior with 3 modes gave very much needed extra flexibility. The model was trained on the first 24 hours of data,

Model	AUC			Log. **p**			TN	FP	FN	TP
	I	II	III	I	II	III				
a	0.90	0.91	0.88	-813	-847	-1231	0.97	0.00	0.03	0.00
b	0.93	0.94	0.91	-1778	-1936	-3127	0.86	0.11	0.01	0.02
c	0.94	0.94	0.91	-3222	-3486	-5773	0.77	0.20	0.00	0.03
d	0.95	0.95	0.92	-3105	-3406	-5685	0.80	0.17	0.00	0.03
e	0.95	0.95	0.92	-3337	-3337	-5500	0.80	0.17	0.00	0.03

Table 4.6.: Results for models from Table 4.5; average value s obtained on the `ThreeGraphFamiliesDataset` dataset. AUC and log-likelihood are calculated for all three types of graphs. log-likelihood isn't normalised as all graphs have the same 100×100 dimension.

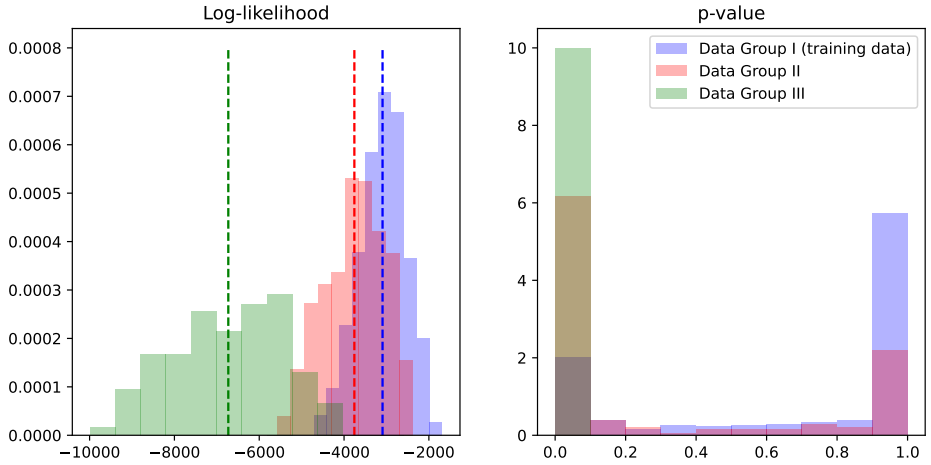

Figure 4.7.: Logarithmic probability of the true adjacency matrix given the model predictive probabilities for graphs in `ThreeGraphFamiliesDataset`

constructing the dataset as explained in subsection 2.2.1, and using anomaly scores from the Pitman-Yor hierarchical model from part 1 as features, together with node in-degree and out-degree. We used three aggregation time intervals, 5 minutes, 15 minutes and 30 minutes, and always 4 time steps, hence spanning between 20 minutes and 2 hours for each sub-graph. After training, the model has been deployed for each source node in correspondence to its most anomalous period according to part 1, to calculate its likelihood or p-value, and yielding results shown in Table 4.7 or Table 4.8 respectively. A visualization of the predicted adjacency matrix is given for one example in Figure 4.8.

The first observation from Table 4.7 and Table 4.8 is that the choice of aggregation time is impactful, as one might expect from the fact that its adequacy might depend upon the anomaly time scale and type of dataset. In the case of this synthetic dataset, the simulated intrusion takes only 6 seconds to travel from the first to target machines, and

	1	1+2 (5min)	1+2 (15min)	1+2 (30min)
N. of anomalies in top 10	7	9	6	7
N. of anomalies in top 20	11	13	10	7

Number of true anomalous source nodes in top 10 and top 20

	1	1+2 (5min)	1+2 (15min)	1+2 (30min)
Wilcoxon p vs. no anomaly	0.968	0.967	0.854	0.760
Wilcoxon p vs. reference	*	0.221	0.554	0.667

Wilcoxon test p-value calculated for ranking of all truly anomalous source nodes, against the same methodology on a dataset without anomaly and against other methods

Table 4.7.: Result for dataset 1, obtained with hierarchical Pitman-Yor, using either only anomaly detection part 1 or 1 and 2 together, with different aggregation times. This table shows results obtained by combining graph likelihood from part 2 with Schulze's method

	1	1+2 (5min)	1+2 (15min)	1+2 (30min)
N. of anomalies in top 10	7	10	10	7
N. of anomalies in top 20	11	12	12	11

Number of true anomalous source nodes in top 10 and top 20

	1	1+2 (5min)	1+2 (15min)	1+2 (30min)
Wilcoxon p vs. no anomaly	0.968	0.964	0.933	0.963
Wilcoxon p vs. reference	*	0.083	0.999	0.942

Wilcoxon test p-value calculated for ranking of all truly anomalous source nodes, against the same methodology on a dataset without anomaly and against other methods

Table 4.8.: Result for dataset 1, obtained with hierarchical Pitman-Yor, using either only anomaly detection part 1 or 1 and 2 together, with different aggregation times. This table shows results obtained by combining p-values from part 1 and 2 using Fisher's method

it is reasonable to think that excessively extended aggregation periods might mask the anomaly. 5 minutes aggregation performs best with either sub-graph ranking options, while 30 minutes aggregation works worst. Between using graph likelihood Table 4.7 and p-values Table 4.8 for ranking anomalous sub-graph, better top-10 and top-20 results are found with the latter for all aggregation times and also comparing against baseline. The sub-graph p-values calculated on the anomaly-free dataset were all very close to 1, meaning that the combination of such p-values with the ones from the first part of anomaly detection left the ordering almost unvaried. This is also confirmed by Wilcoxon's test: using p-values instead of likelihoods led to one of the best results, with aggregation 5 minutes, with Wilcoxon's p-value ~ 0.08. We can conclude that, in the simulated case, the second part of anomaly detection proved successful, caveat the selection of aggregation time. 5 minutes aggregation worked best according to all metrics.

4.2. LANL Dataset

The LANL dataset comprises 58 days of network activity. To fit within time constraints, the dataset used in this project is reduced to the first 42 hours for training an up to the first 240 hours for deployment. With such reduction, the computation for the first part of anomaly detection would take between 1.5 days and 4 days depending on the modelling approach (Pitman-Yor, streaming Pitman-Yor, etc...), with the exception of the impractical Poisson marked point process which run for more than 13 days before being stopped, confirming that this approach is inapplicable in practice. Training of the DynVAE model, which has scope for performance optimizations, is expected to take a maximum of 50 days or less, depending on the patience threshold being hit. Due to time constraints, the model training was stopped after 10 days. For clarity, such long training times are not critical when deploying the model, as inference is very fast. The choice of what subset of the full LANL dataset to use was made to retain two know anomalies, source nodes **C17693** and **C22409**, out of a total 16230 source nodes and 15417 destination nodes.

4.2.1. First Part of Anomaly Detection

The anomaly ranking for the two known anomalies in the selected sub-set of the LANL dataset are shown in Table 4.9; the distance-dependent CRP proved to be the best model, gaining several positions for the truly anomalous nodes. Streaming Pitman-Yor performed worst, both with 12 and 36 hours window: this shows a pitfall of such method, the performance of which is affected by disregarding older observations. Longer windows would produce results more and more similar to the standard Pitman-Yor, but, anyway, the influence of the chosen window on the result is a potential liability of the approach. It is difficult to conclude on the comparison between Pitman-Yor and Dirichlet approaches, which perform one better than the other on either anomaly. Figure 4.9 shows QQ plot of the p-values, which should be uniformly distributed under a correct model specification. However, we know that at least two nodes are anomalous, hence we expect the p-values

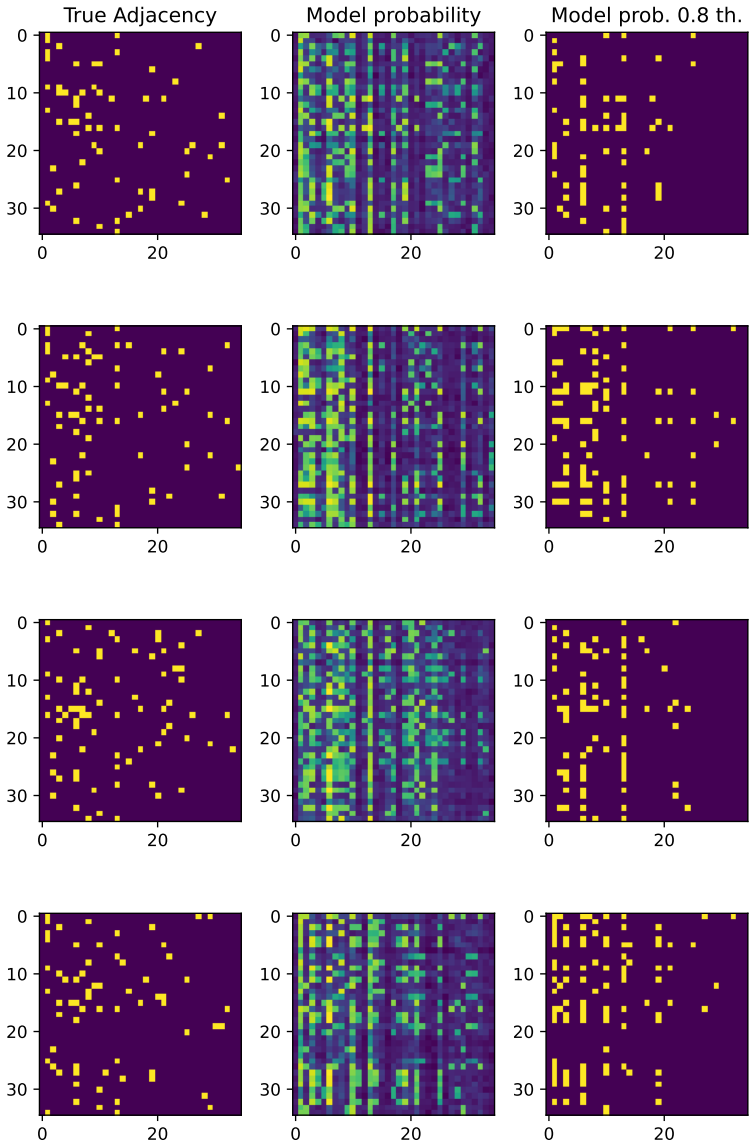

Figure 4.8.: Example application of trained DynVAE model to dynamic graph; four time steps from top to bottom.

to be uniformly distributed only in first approximation. We can confirm, also from this point of view, that DDCRP is the best performing model.

	PY	DP	DDCRP	STR. PY (12h)	STR. PY (36h)
C17693	332	304	81	1522	1182
C22409	435	581	398	1610	1010

Table 4.9.: Ranking of known anomalous source nodes according to model specification.

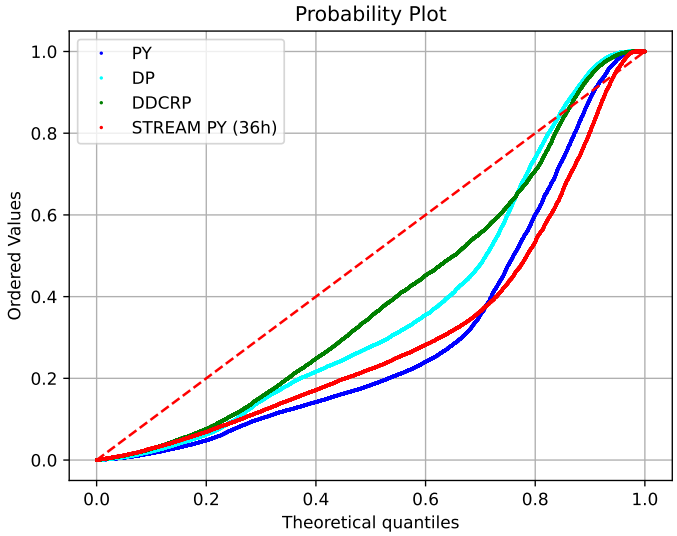

Figure 4.9.: Anomaly detection part one p-value quantiles, against theoretical uniformly distributed quantiles

4.2.2. Second Part of Anomaly Detection

The same DynVAE model used for the synthetic dataset has been trained on the first 42 hours of the LANL dataset, using 30 minutes aggregation intervals and Pitman-Yor scores from the local anomaly detection. As anticipated, training of DynVAE on the LANL dataset had to be terminated earlier, due to time constraints. Figure 4.10 shows the the training history: at the end of training, the AUC is higher than 0.82 for all time-steps, and the average precision score is higher than 0.75. Looking at the NLL trend, the optimization was still improving when stopped. Relative AUCs between time-steps also show that, towards the end of training, the graph recurrent neural network is improving, as all AUCs get closer to each other. As the second part of anomaly detection is auxiliary to the first part of anomaly detection, we considered only potentially anomalous nodes flagged with p-value smaller than 0.03 by the first part: 2000 nodes out of 13000. Monte Carlo simulations to infer the adjacency matrix and calculate p-values took approximately $30s$ per node on the DELL laptop with specifications as mentioned before (importantly, this time is quoted for CPU). Inference is completely parallelisable: on a 128 cores machine, calculating the results should take only 8 minutes. There is also scope to optimise the code for performance. The resulting final ranking, obtained

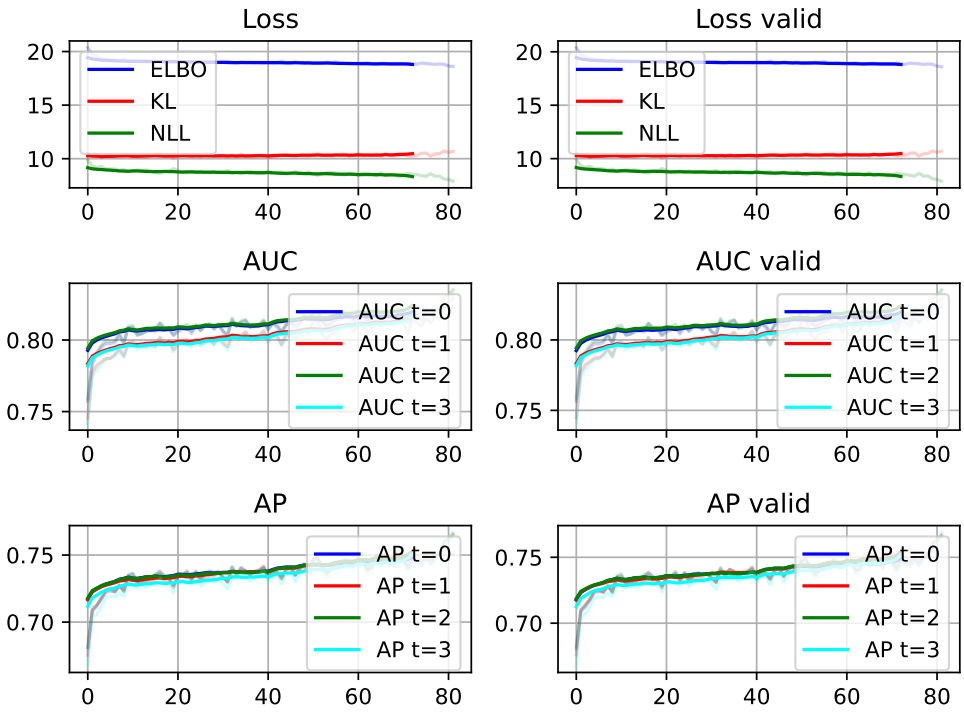

Figure 4.10.: DynVAE training history on LANL dataset

combining results from both anomaly detections, is shown in Table 4.10. There is a dramatic improvement for **C17693**, while **C22409** is ranked lower than with the sole local anomaly detection, even if by a relatively low margin. Not having improved on both machines makes it difficult to conclude firmly, and there are some considerations to make:

- Model training was terminated prematurely for time constraints; although with already good metrics, there is potential for improvement;
- It was not possible to construct the dynamic graph for about 100 nodes out of 2000, either because they interacted with less than 10 nodes in the 2 hours preceding the potential anomaly, or because there were no communications for more than 30 minutes in the 2 hours preceding the potential anomaly. Impossibility to construct dynamic graphs of the postulated form should be carefully addressed;
- The real dataset, as opposed to the synthetic dataset, is rich and varied: increased model dimension or using multiple time aggregations could be attempted to allow for extra flexibility;
- The p-value distribution for the second part of anomaly detection is far from uniform: 87% of p-values are in $[0.98, 1.00]$, 7% in $[0.00, 0.02]$ and the remaining

	Part 1 p-value	Part 2 p-value	Combined	Ranking
C17693	0.0026	0.0000	0.0000	19
C22409	0.0036	1.0000	0.0238	519

Table 4.10.: Ranking of known anomalous source nodes using a Pitman-Yor hierarchical process for the first part of anomaly detection and 30 minutes aggregation time for the second part of anomaly detection, expressing the result of the DynVAE anomaly detection with p-values as these performed best on the synthetic dataset.

6% is roughly uniformly distributed in $[0.02, 0.98]$. Moreover, 70 p-values out of 1900 defined p-values are exactly 0, and 940 are exactly 1; this makes p-values combination impossible with most p-value combination methods. In our case we ordered nodes with p-value 0 by using only the score from the first part of anomaly detection.

A cautions conclusion is that there is not enough evidence to promote the second part of anomaly detection on the LANL dataset based on this test, although there is potential for improvement.

5. Conclusion

This project examined the possibility of extending previous work on local anomaly detection in computer networks, based on Dirichlet and Pitman-Yor processes, by dropping the exchangeability assumption in three different ways. The first leverages modelling of the stochastic time sequence of events with a Poisson point process; this introduces a link between arrival times and sequence of realized connections that makes the description no longer exchangeable, however, at high computational cost. The second approach is a streaming version of the Pitman-Yor process, which disregards older observations depending on a defined time window. The third and last approach uses a distance-dependent Chinese restaurant process DDCRP to introduce dependence on inter-arrival times, with exponential decay. All methods were compared and contrasted on two synthetic datasets, based on different generating processes, with simulated intrusion; the performance of each method is expressed in terms of ranking of true anomalies in the top-10 and top-20, as well as the overall prioritization of true anomalies based on Wilcoxon's signed-rank test. On the first synthetic dataset, generated using a Pitman-Yor hierarchical process, the proposed streaming Pitman-Yor and DDCRP methodologies perform as well, or slightly better, than the benchmark Pitman-Yor model from literature. On the second dataset, generated with a hierarchy of Pitman-Yor and DDCRP processes, DDCRP and benchmark Pitman-Yor approaches work best. Good performance of the DDCRP approach is also found with the Los Alamos National Laboratory dataset, where the true anomalous nodes gain a significant number of positions in the anomaly rank with respect to benchmark methods. On the other hand, the Pitman-Yor streaming version is quite poor, which highlights the challenge of defining a meaningful streaming window. We can conclude that the distance dependent Chinese restaurant process formulation of the problem was successful, and a good challenger of state-of-art methods based on the synthetic and real datasets used in this work.

The methodology described so far could be regarded as local, as it expresses a level of "surprise" based on the history of connections from a certain node. In an attempt to reduce false alarms, we also wish to verify if the communication patterns in the neighbourhood of a potentially anomalous node are unlikely with respect to the type of activity seen in the entire network, a process that could be regarded as global. One possibility is to consider dynamic sub-graphs of the entire network in the neighbourhood of the potentially anomalous node, considering historical data within a certain time interval. The novel contribution in this work is towards using a dynamic graph variational auto-encoder to learn sane patterns within the entire network and deploy the model on unseen, potentially corrupted graphs by integrating over the latent space, with Monte Carlo methods, to calculate their likelihood or p-value given the model. The hope to spot dynamic anomalies is backed by the fact that most cyber intrusions are charac-

terised by traversal movement between corrupted and target machines, a very unlikely pattern in normal conditions. Scores from the first part of anomaly detection are used as node features as well as a measure to select which nodes to consider for the second part, viewable as a supplement to the first. Exploratory graph classification experiments were run to choose appropriate convolution layers for our model and successfully confirm, in a simple setting, the discriminatory power of an approach that seeks to use the variational auto-encoder trained on a class of graphs, to discriminate from other and even between other classes. The potential of this approach was also shown on both synthetic and real datasets, basing the simulations on Pitman-Yor p-values from the first part of anomaly detection. In order to construct the dynamic graphs, data were aggregated by intervals of 5, 10 or 30 minutes on the synthetic dataset, and 30 minutes on the LANL dataset. The effectiveness of the proposed approach was demonstrated on the synthetic dataset, especially with 5 and 10 minute aggregations, where up to 3 more true anomalies were ranked in the top 10. On the Los Alamos National Laboratory dataset, the additional methodology gained 300 positions to one known anomaly and lost 100 to the other, making it more difficult to conclude on performance. However, there is scope to improve, as model training was terminated prematurely due to time constraints and hyper-parameter tuning is almost surely needed given the richness and variety within the real dataset. One challenge with the proposed approach is in the choice of the aggregation interval to construct dynamic graphs, a choice that in practice might require to train different models for a set of different aggregations and use them altogether.

In the introduction we mentioned two requirements in cybersecurity modelling: necessity to reduce false alarms to distress security teams and necessity to react virtually in real time. The former was addressed by introducing a second part of anomaly detection. The second requirement is also satisfied: the DDCRP-based model took approximately as long as the benchmark Dirichlet and Pitman-Yor models to calculate sequentially, while inference for the variational auto-encoder took roughly 30s on CPU. In applicative settings, most calculations for the first part of anomaly detection can be federated or distributed, while inference for the second part of anomaly detection can be parallelised, intuitively reducing computation times to acceptable levels.

The main limitations of the approaches proposed for the first, local part of anomaly detection where computation time for the inhomogeneous Poisson process, performance for the streaming version of the Pitman-Yor process; having a streaming window seems to create too much of an abrupt forgetting, which future work might try and address by using a forgetting factors version of the process [25] instead. We did not find particular limitations with the distance-dependent CRP; given the potential unleashed by including an explicit dependence on inter-arrival times, further exploration of dependent processes [20, 32, 19, 9] could be considered in future work, albeit the distance-dependent CRP seems the most convenient approach computationally. The main limitation of the second part of anomaly detection is the choice of aggregation times to construct the dynamic graphs; this aspect could be further explored in future work, as knowledge on the types of attacks might be necessary to fully quantify the potential of the proposed approach. Potential that would also benefit from longer training and VAE hyper-parameter tuning, which were not possible on the real dataset due to time constraints.

A. Dynamic Graphs Global Methods

Within global methods for dynamic graph modelling, we can make a distinction between **spectral**, **distance-based** and **factorisation** methods. Spectral methods are often used under the random dot product graph generative assumption [2, 27, 6] and are based on the eigen-decomposition of the adjacency matrix A or the Laplacian matrix $L = D^{-1/2}AD^{-1/2}$, where D is the diagonal degree matrix. A random dot product graph is defined as follows:

Definition A.0.1 (Random dot product graph)**.** Let the adjacency matrix $A \in \mathbb{R}^{n\times n}$ and let $\mathcal{X}_d$ be a subspace of the Euclidean space $\mathbb{R}^d$ such that $\forall x, y \in \mathcal{X}_d$ we have that $x^T y \in [0, 1]$. Let the latent vector $x_i \in \mathcal{X}_d$ for each node be random variables distributed according to $\mathcal{F}$ with support $\mathcal{X}_d$. If the matrix $X \in \mathbb{R}^{n\times d}$ has the latent vectors as rows, then the matrix of connection probabilities is $P = XX^T$ and the entries of the adjacency matrix A_{ij} are Bernoulli random variables with probability P_{ij}. Mathematically:

$$A_{ij} \overset{ind}{\sim} Bernoulli(x_i^T x_j) \tag{A.19}$$

Spectral decomposition allows to select a lower-dimensional representation of the graph by retaining d scaled eigenvectors corresponding to the d highest eigenvalues, which can be used, for example, for clustering with k-means. [2] provide a survey of such methods - in the static graph setting - and highlight some challenges related to uniquely identifying the positions due to invariance to translation and rotation, and the possibility to retain only positive eigenvalues. [27] propose a solution to allow for disassortativty (negative eigenvalues) by modelling $A_{ij} \overset{ind}{\sim} Bernoulli(x_i^T I_{qp} x_j)$ (cf. A.19) where $\mathbf{I}_{qp}$ is a diagonal block matrix with q ones and p minus ones. This extension made the approach suitable for cybersecurity, where transitivity and assortativity are rare; in fact, similar computers do not necessarily communicate, and triangular communications are rare too, especially in intrusion scenarios. Dynamic extensions to the static spectral embedding either exploit the projection of all graphs $(\mathcal{V}_t, \mathcal{E}_t)$ onto the same latent space, a practice called **graph omnibus**, or independently project each graph $(\mathcal{V}_t, \mathcal{E}_t)$. As explained by [6], there are two stability properties that we would like a dynamic graph embedding to fulfil: nodes behaving similarly should have close-by positions at any given time, and nodes behaving similarly across time should have a constant position. The authors show how **omnibus embedding** and **independent spectral embedding** fail to satisfy either of the two, while a procedure called **unfolded spectral embedding** has these desirable properties:

Definition A.0.2 (Unfolded spectral embedding)**.** For a sequence of adjacency matrices $A_t \in \mathbb{R}^{n\times n}$, $t = 1, \cdots, T$, **unfolded spectral embedding** uses singular value decom-

position of the matrix $A = (A_1|\cdots|A_T) \in \mathbb{R}^{n\times nT}$ to obtain $A \simeq XY^T$ with the right embedding $Y \in \mathbb{R}^{nT\times d}$ containing the desired representations as rows.

In the area of distance-based models, [13] work with social networks and model the binary-valued adjacency matrix with entries A_{ij} by expressing the probability of a link realisation $P(A_{ij}|z_i, z_j, x_{ij}, \theta)$, where $z_i \in \mathbf{R}^k$ are latent position vectors, x_{ij} are vector-valued reciprocal covariates and θ is a set of parameters, with a logistic regression, and log odds dependent on the distance between the latent vectors:

$$\eta_{ij} = logodds(A_{ij} = 1|z_i, z_j, x_{ij}, \alpha, \beta) = \alpha + \beta^T x_{ij} - |z_i - z_j|$$

or their dot product:

$$\eta_{ij} = logodds(A_{ij} = 1|z_i, z_j, x_{ij}, \alpha, \beta) = \alpha + \beta^T x_{ij} + \frac{z_i^T z_j}{|z_j|}$$

Reciprocity and **transitivity** are embedded in this formulation, although the latter is relaxed when the log odds depend on dot products. The challenges, discussed by the authors, are identifying the correct dimension of the latent space and solving for the positions. Another approach, popular for link prediction, is **Poisson matrix factorisation**: [24] model the number of purchases in a recommender system (could also be the number of connections in a computer network) in a bipartite graph like $N_{ij} \sim Pois(\alpha_i \cdot \beta_j)$, where α_i are user-specific latent features and β_j are host-specific latent features. This approach to link prediction proved particularly suitable for applications with sparse matrices, like in cybersecurity, and provided a natural way to handle cold starts [1], it being based on latent features known a priori. Another methodology that can be regarded as global, despite the local flavour, is presented in [23]: the counting process for the number of edge-level connections $N(t)$ is described by a self-exciting process with rate $\lambda(t)$ conditioned on the history up to time t:

$$\lambda(t) = \lambda + \sum_{k>N(t)-r}^{N(t)} \omega(t - t_k)$$

where ω is a non-increasing, non-negative excitation function. If $r = 1$ we have a Markov-like process, with $r = \infty$ we have a Hawkes process, finally with $r = 0$ we have a Poisson process. In the paper, the rate of update is conditioned upon latent node vectors, a feature that gives a global flavour to the approach and allows to handle **cold starts**, as node features are known a priori.

[1] In general, considerations about the model's ability to handle unseen observations in a streaming context is relevant to cybersecurity applications

B. Pitman-Yor p-value calculation

There are some simplification in the calculations of the Pitman-Yor process p-value 2.3 that make it extremely tractable. First of all, we will be working with a uniform discrete base measure, meaning that $p_0(x) = p_0$, hence asking if $p(x|\mathbf{x}) \leq p(x_{n+1}|\mathbf{x})$ is equivalent to asking if $N_{in} \leq N_{jn}$ when $x \neq x_{n+1}$. We simplify the notation for N_{in} as $n(x)$, indicating the number of occurrences of x up to n observations. We have that:

$$\mathbf{p}(x_{n+1}) = \sum_{n(x)\leq n(x_{n+1})} \frac{(\alpha + dK_n)p_0 + n(x) - d\mathbb{I}_{(0,\infty)}(n(x))}{\alpha + n} = \frac{\left(\sum_{n(x)\leq n(x_{n+1})} 1\right)(\alpha + dK_n) + \sum_{n(x)\leq n(x_{n+1})} n(x) - d\sum_{n(x)\leq n(x_{n+1})} \mathbb{I}_{(0,\infty)}(n(x))}{\alpha + n}$$

When x_{n+1} is unseen, the only x such that $n(x) \leq n(x_{n+1})$ are the other unseen xs. The probability x unseen is $p(x|\mathbf{x}) = \frac{(\alpha+dK_n)p_0}{\alpha+n}$, hence:

$$\mathbf{p}_{unseen} = \sum_{x\in\mathcal{V}|n(x)=0} \frac{(\alpha + dK_n)p_0}{\alpha + n} = \frac{\alpha + dK_n}{\alpha + n}\frac{|\mathcal{V}| - K_n}{|\mathcal{V}|} \underset{|\mathcal{V}|\to\infty}{\to} \frac{\alpha + dK_n}{\alpha + n} \tag{B.20}$$

For already seen values, it is a lot easier to calculate the p-value based on the observations with $n(x) > n(x_{n+1})$:

$$\mathbf{p}(x_{n+1}) = 1 - \frac{\left(\sum_{n(x)>n(x_{n+1})} 1\right)\frac{(\alpha+dK_n)}{|\mathcal{V}|} + \sum_{n(x)>n(x_{n+1})} n(x) - d\sum_{n(x)>n(x_{n+1})} 1}{\alpha + n} \underset{|\mathcal{V}|\to\infty}{\to} 1 - \frac{\sum_{n(x)>n(x_{n+1})} n(x) - d\sum_{n(x)>n(x_{n+1})} 1}{\alpha + n} \tag{B.21}$$

The same procedure can be done for destination nodes and then, conditional on destination, for source nodes.

C. Graph Convolution Layers

This subsection summarizes some of the core ideas from [41], which provides an extensive overview of graph convolutional layers. Some of the challenges of graph neural networks, also GNNs, is that graphs are invariant to labelling. In other words, graphs apparently dissimilar in their adjacency matrices might in fact be equivalent under relabelling of the nodes. This is one of the main reasons for which specific layers have been developed for graphs. GNNs broadly follow a "recursive neighbourhood aggregation" also known as **message passing** scheme, where each node inherits its feature vector from an aggregation of the feature vectors of its neighbours. After k message passes, a node feature vector retains information about its neighbourhood up to path distance k. Different neighbourhood aggregations and graph-level pooling schemes have been proposed, and in [41] their performance is compared and contrasted, backed by the Weisfeiler-Lehman graph isomorphism test.

Given the **edge set** $\mathcal{E}$ and **node feature vectors** $\mathbf{x}_i$, the target is to learn a representation vector $\mathbf{h}_i$ for each node (or for the entire graph $\mathbf{h}_G$). The k-th layer of a message passing algorithm is:

$$a_i^{(k)} = AGGREGATE^{(k)}\left(\{h_j^{(k-1)} : x_j \in \mathcal{N}(x_i)\}\right)$$

$$h_i^{(k)} = COMBINE^{(k)}\left(h_i^{(k-1)}, a_i^{(k)}\right)$$

where $h_i^{(k)}$ is the feature vector, initialised as $h_i^{(0)} = \mathbf{x}_i$ and $\mathcal{N}(x_i)$ is a set of nodes adjacent to x_i. The choice of the aggregation and combination functions is crucial. In **GraphSAGE**:

$$a_i^{(k)} = MAX\left(\{ReLU(W \cdot h_j^{(k-1)}), \forall x_j \in \mathcal{N}(x_i)\}\right)$$

where W is learnable matrix, MAX is element-wise max pooling, and the concatenation is followed by a linear mapping:

$$W \cdot \left[h_i^{(k-1)}, a_i^{(k)}\right]$$

In **Graph Convolutional Networks**, GCNs, mean pooling is used instead, and aggregation and combination are integrated in one step:

$$h_i^{(k)} = ReLU\left(W \cdot MEAN\{h_j^{(k-1)}, \forall x_j \in \mathcal{N}(x_i) \cup \{x_i\}\}\right)$$

For graph classification, the $READOUT$ function aggregates node features from the

final iteration, to obtain the entire graph's representation h_G:

$$h_G = READOUT(\{h_i^{(K)} : x_i \in G\})$$

The authors proceed by delineating a theoretical framework for a maximally powerful GNN. The **Weisfeiler-Lehman test** quantifies whether two graphs are topologically identical by iteratively aggregating the labels of nodes and their neighbours and hashing the aggregated labels into unique new labels. Two graphs are non-isomorphic if at some iteration the labels of the nodes between the two graphs differ. Based on this test, [30] proposed the Weisfeiler-Lehman subtree kernel that measures the similarity between graphs. Let us assume the input features are countable, for example in $\{a, b, c, \cdots\}$. Feature vectors of neighbouring nodes form a multi-set, as the same feature can appear multiple times. A maximally powerful GNN maps two nodes to the same location only if they have identical subtree structures and neighbouring nodes features. In other words, similar neighbourhoods should be mapped to similar embeddings, and different neighbourhoods to different embedding: the function should be injective. We would also like isomorphic graphs to have the same representation, and non-isomorphic graphs to have different representations.

Lemma C.0.1 (Most powerful GNNs)**.** A graph neural network is at most as powerful as a WL test

The core of [41] is the proposal of a new graph convolutional layer as powerful as the WL test:

Theorem C.0.2 (Powerful GNNs construction)**.** There exists a GNN as powerful as the WL test if the GNN aggregates and updates node features iteratively with:

$$h_i^{(k)} = \phi\left(h_i^{(k-1)}, f\left(\left\{h_j^{(k-1)} : x_j \in \mathcal{N}(x_i)\right\}\right)\right)$$

for ϕ and f injective, and if the graph-level readout is also injective.

The article focuses on **countable feature sets**, as uncountable sets and distances characterisation in the learned space are left for future work. If we assume that $\mathcal{X}$ is countable, then there exists a function f so that $\sum_{x \in X} f(x)$ is unique for each multi-set $X \subset \mathcal{X}$. Moreover, any multi-set function g can be decomposed as $g(X) = \phi\left(\sum_{x \in X} f(x)\right)$ for some function ϕ. Note that functions that are injective on sets, might not be on multi-sets, like the mean aggregator, and the target here is to build a function that is injective on multi-sets indeed. Of the possible implementations, the authors propose the following **G**raph **I**somorphism **N**etwork:

$$h_i^{(k)} = MLP^{(k)}\left((1 + \epsilon^{(k)})h_i^{(k-1)} + \sum_{x_j \in \mathcal{N}(x_i)} h_j^{(k-1)}\right) \tag{C.22}$$

In the first iteration the MLP is not needed if the input features are one-hot encodings, as their summation alone is injective. It is important to note that 1-layer MLPs are

like generalised linear models and not enough to distinguish between multi-sets, hence it is desirable to use more than one layer. We will later compare the performance of SageCONV, GCN and GIN on our dataset, but in principle we seek to use the maximally powerful GNN in our application, where being able to discriminate between non isomorphic graphs is crucial.

D. Graph Variational Auto-encoder

The variational auto-encoder [14, 26] is a latent variable generative model, and is a prescribed or likelihood-based model, as each observation $x \in \mathbb{R}^d$ is prescribed with a noise model $p_\theta(x|z)$ given the latent variable $z \in \mathbb{R}^l$. The model is $p_\theta(z)p_\theta(x|z)$ and the marginal likelihood of a single observation is:

$$p_\theta(x) = \int_z p_\theta(z)p_\theta(x|z)dz \tag{D.23}$$

This is already the key to adopting such approach in our work: given a model trained on "normal" graphs $p_\theta(\cdot|z)$ we seek to calculate, with Monte Carlo sampling, the probability of a new observation like in Equation D.23.

In relation to model training, however, this integral is intractable, and the technique is to resort to the Evidence Lower Bound using a parametrised distribution $q_\phi(z|x)$ of our choosing:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}[p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right) \tag{D.24}$$

We can view $q_\phi(z|x)$ as the **encoder**, while $p_\theta(x|z)$ is the **decoder**, and the Kullback-Leibler term is there to ensure closeness to the pre-defined prior distribution. The first term is also called **reconstruction loss**, while the second term is a regularisation term that ensures that the encoded distribution does not diverge too much from the prior distribution.

The concept of VAEs has been applied to graphs, for example in [15] to undirected graphs, by using a two-layer GCN encoder and a simple inner product decoder. Given a graph $\mathcal{G}$, with adjacency matrix $A \in \mathbb{R}^n \times \mathbb{R}^n$ and degree matrix D, latent variables $z_i \in \mathbb{R}^l$ and node features $x_i \in \mathbb{R}^d$, the encoder is:

$$q_\phi(Z|X,A) = \prod_{i=1}^{n} q_\phi(z_i|X,A) \;\text{ with }\; q_\phi(z_i|X,A) = \mathcal{N}(z_i|\mu_i, diag(\sigma_i^2))$$

The matrix of mean vectors μ_i is calculated as $\mu = GCN_\mu(X,A)$ and similarly $\log\sigma = GCN_\sigma(X,A)$, where GCN_μ and GCN_σ are two-layer GCN which share the first layer. The generative model uses drawn vectors z_i and z_j to reconstruct the adjacency matrix:

$$p_\theta(A|Z) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(A_{ij}|z_i,z_j) \;\text{ with }\; p(A_{ij}=1|z_i,z_j) = \sigma(z_i^T \cdot z_i)$$

The parameters are optimised by minimising Equation D.24 with Gaussian prior $p(Z) = \prod_{i=1}^{n} p(z_i) = \prod_{i=1}^{n} \mathcal{N}(z_i|0,\mathbb{I}_n)$. For very sparse adjacency matrices, it is beneficial to

re-weight terms with $A_{ij} = 1$ or alternatively sub-sample terms with $A_{ij} = 0$.

Bibliography

[1] C. E. Antoniak. Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems. *The Annals of Statistics*, 2(6):1152–1174, Nov. 1974.

[2] A. Athreya, D. Fishkind, K. Levin, V. Lyzinski, Y. Park, Y. Qin, D. Sussman, M. Tang, J. Vogelstein, and C. Priebe. Statistical inference on random dot product graphs: A survey. *Journal of Machine Learning Research*, 18, 09 2017.

[3] S. Choudhury, L. B. Holder, G. Chin Jr., K. Agarwal, and J. Feo. A selectivity based approach to continuous pattern detection in streaming graphs. *CoRR*, abs/1503.00849, 2015.

[4] CrowdStrike. Lateral movement. `https://www.crowdstrike.com/cybersecurity-101/lateral-movement/`, 2023. Last accessed 23/08/01 at 09:49.

[5] P. I. Frazier D. M. Blei. Distance dependent chinese restaurant processes. *Journal of Machine Learning Research*, 12(2011):2461–2488, 2011.

[6] I Gallagher, A. Jones, and P. Rubin-Delanchy. Spectral embedding for dynamic networks with stability guarantees. *ArXiv*, abs/2106.01282, 2021.

[7] B. Gold, D. Curwin, and C. McClister. Understand and investigate Lateral Movement Paths (LMPs) with Microsoft Defender for Identity. `https://learn.microsoft.com/en-us/defender-for-identity/understand-lateral-movement-paths`, 2023. Last accessed 23/08/01 at 09:47.

[8] B. Gold, D. Curwin, C. McClister, S. Sagir, R. Wiselman, and M. Baldwin. Lateral movement alerts. `https://learn.microsoft.com/en-us/defender-for-identity/lateral-movement-alerts`, 2023. Last accessed 23/08/01 at 09:48.

[9] J. E. Griffin and M. F. J. Steel. Order-based dependent dirichlet processes. *Journal of the American Statistical Association*, 101(473):179–194, 2006.

[10] E. Hajiramezanali, A. Hasanzadeh, N. Duffield, K. R. Narayanan, M. Zhou, and X. Qian. Variational graph recurrent neural networks. *CoRR*, abs/1908.09710, 2019.

[11] I. Hawryluk, H. Hoeltgebaum, C. Sodja, T. Lalicker, and J. Neil. Peer-group Behaviour Analytics of Windows Authentications Events Using Hierarchical Bayesian Modelling. *arXiv*, 2209.09769, 2022.

[12] N. Heard and P. Rubin-Delanchy. Network-wide anomaly detection via the dirichlet process. In *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, pages 220–224, 2016.

[13] P. D. Hoff, A. E. Raftery, and M. S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

[14] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv*, 1312.6114, 2022.

[15] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv*, 1611.07308, 2016.

[16] R. Kulhavy and M. B. Zarrop. On a general concept of forgetting. *International Journal of Control*, 58(4):905–924, 1993.

[17] P. J. Laub, T. Taimre, and P. K. Pollett. Hawkes processes. *arXiv*, 1507.02822, 2015. [math, PR].

[18] J. Liu, A. Kumar, J. Ba, J. Kiros, and K. Swersky. Graph normalizing flows. *arXiv*, 1905.13177, 2019.

[19] S. N. MacEachen. Dependent dirichlet processes. *Technical Report*, 2000. Department of Statistics, The Ohio State University.

[20] P. Orbanz and Y. W. Teh. Bayesian nonparametric models. In *Encyclopedia of Machine Learning*, 2010.

[21] F. S. Passino. *Big Data: Statistical scalability with PySpark*. Lecture notes, unpublished Material, 2022. 138-139.

[22] F. S. Passino. *Big Data: Statistical scalability with PySpark*. Lecture notes, unpublished Material, 2022. 143-146.

[23] F. Sanna Passino and N. A. Heard. Mutually exciting point process graphs for modeling dynamic networks. *Journal of Computational and Graphical Statistics*, 32(1):116–130, sep 2022.

[24] F. Sanna Passino, Melissa J. M. Turcotte, and N. A. Heard. Graph link prediction in computer networks using Poisson matrix factorisation. *The Annals of Applied Statistics*, 16(3):1313 – 1332, 2022.

[25] A. Quinn and M. Karny. Learning for non-stationary Dirichlet processes. *International Journal of Adaptive Control and Signal Processing*, 21:827 – 855, 12 2007.

[26] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv*, 1401.4082, 2014.

[27] P. Rubin-Delanchy, J. Cape, M. Tang, and C. E. Priebe. A statistical interpretation of spectral embedding: the generalised random dot product graph. *arXiv*, 1709.05506, 2021.

[28] F. Sanna Passino and N. Heard. Modelling dynamic network evolution as a Pitman-Yor process. *Foundations of Data Science*, 3(1):293–306, 2019.

[29] M. Schulze. The schulze method of voting. *CoRR*, abs/1804.02973, 2018.

[30] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

[31] K. Sigman. *Batched Point Processes*. School of Operations Research and Industrial Engineering, 2007.

[32] Mark F. J. Steel. Bayesian time series analysis. *The New Palgrave Dictionary of Economics*, 2008.

[33] Microsoft Defender Research Team. Cyberbattlesim. `https://github.com/microsoft/cyberbattlesim`, 2021. Created by C. Seifert, M. Betser, W. Blum, J. Bono, K. Farris, E. Goren, J. Grana, K. Holsheimer, B. Marken, J. Neil, N. Nichols, J. Parikh, H. Wei.

[34] Y. W. Teh. Dirichlet process. `https://www.stats.ox.ac.uk/~teh/research/npbayes/Teh2010a.pdf`.

[35] M. Turcotte, N. Heard, and J. Neil. Detecting localised anomalous behaviour in a computer network. In H. Blockeel, M. van Leeuwen, and V. Vinciotti, editors, *Advances in Intelligent Data Analysis XIII*, pages 321–332, Cham, 2014. Springer International Publishing.

[36] M. J. M. Turcotte, A. D. Kent, and C. Hash. *Unified Host and Network Data Set*, chapter Chapter 1, pages 1–22. World Scientific, nov 2018.

[37] European Union. Cybersecurity: main and emerging threats. `https://www.europarl.europa.eu/news/en/headlines/society/20220120STO21428/cybersecurity-main-and-emerging-threats`, 2023. Last accessed 21-03-2023 - 10:45.

[38] European Union. Cybersecurity: why reducing the cost of cyberattacks matters. `https://www.europarl.europa.eu/news/en/headlines/society/20211008STO14521/cybersecurity-why-reducing-the-cost-of-cyberattacks-matters`, 2023. Update 23-11-2022 - 14:29.

[39] D. Wang, T. Zhao, N. V. Chawla, and M. Jiang. Dynamic attributed graph prediction with conditional normalizing flows. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1385–1390, 2021.

[40] Zhang Q. Guo D. et al. Wang, X. A survey of continuous subgraph matching for dynamic graphs. *Knowl Inf Syst*, (65):945–989, 2023.

[41] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv*, 1810.00826, 2019.